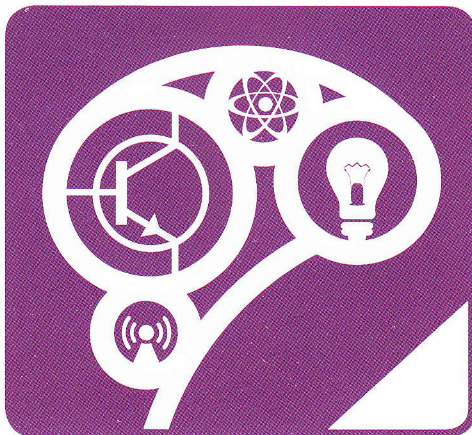
N°10 | JANVIER - FÉVRIER 2016



HACKABLE MAGAZINE

DÉMONTEZ | COMPRENEZ | ADAPTEZ | PARTAGEZ

France MÉTRO. : 7,90 € – CH : 13 CHF – BEL/LUX/PORT.CONT : 8,90 € – DOM/TOM : 8,50 € – CAN : 14 $ CAD

ARDUINO

Créez une horloge électro-vintage originale à cadrans analogiques à aiguille

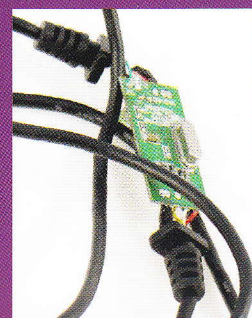
p. 24



GPS

Ajoutez la géolocalisation à vos projets Arduino en utilisant un récepteur GPS

p. 04



CROQUIS

Prenez en charge proprement les interruptions en évitant les bugs

p. 86

REPÈRES

Apprenez à compiler et installer un nouveau noyau pour votre Raspberry Pi

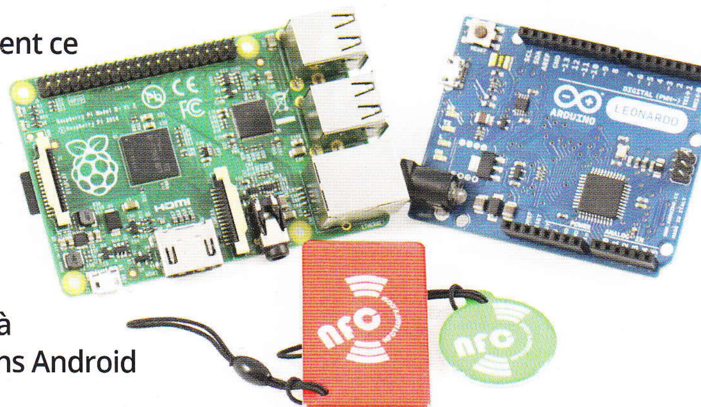
p. 92

NFC & RFID EN PRATIQUE !

sur Arduino & Raspberry Pi

p. 32

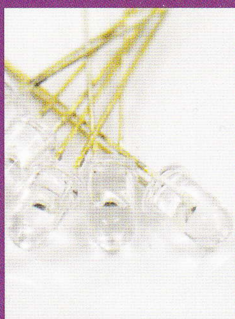
- Comprenez enfin clairement ce que sont NFC et RFID !
- Utilisez les tags NFC avec votre Raspberry Pi
- Lisez et gérez vos tags avec votre Arduino
- Complétez votre trousse à outils avec des applications Android



COMMUNICATION

Faites dialoguer deux Arduino avec une simple paire de leds

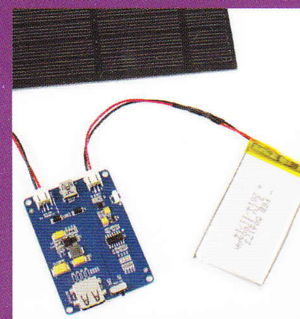
p. 16



ALIMENTATION

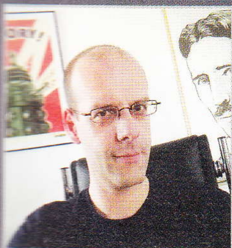
Panneau solaire et autonomie en toute simplicité avec le LiPo Rider Pro

p. 77





ÉDITO



Souvent les choses qui paraissent les plus simples sont parmi les plus compliquées.

Je dirais même que ce qui rend les choses simples pour les uns, les complique souvent pour d'autres. Prenez l'USB par exemple, pour l'utilisateur c'est le rêve du plug'n'play (les anciens utilisateurs Windows 95 esquisseront sans doute un demi-sourire) devenu réalité... ou presque. Mais pour les programmeurs et amateurs d'électronique numérique, cela est bien plus délicat qu'une simple connexion série ou parallèle (là ce sera plutôt un sourire nostalgique).

Il en va de même avec la thématique principale de ce numéro. Les technologies NFC et RFID, souvent résumées sous l'appellation « sans contact », semblent simples d'utilisation, mais sous le capot c'est une tout autre affaire. Avant même de toucher du doigt la mise en pratique, il est indispensable de commencer par savoir de quoi on parle avec précision.

J'ai essayé de condenser cette première étape au mieux en la rendant la plus confortable et digeste possible, mais le coût de l'utilisation de ces technologies, le ticket d'entrée comme disent certains, n'est pas inexistant. Les perspectives cependant sont à la hauteur de l'effort à fournir, car bien entendu, je parle d'un coût en temps et en calories brûlées par nos adorables et espiègles petits (mais nombreux) neurones.

À cœur vaillant rien d'impossible ! Et je ne doute pas que, vaillants, vous devez l'être puisque, après tout, vous êtes lecteur du magazine et me supportez en guise de compagnon ou de guide pour vos explorations électroniques. Je vous invite donc, encore une fois, à me suivre à l'aventure pour découvrir de nouveaux horizons.

Madame, monsieur, je suis Denis, votre rédacteur en chef. L'ensemble de l'équipage a le plaisir de vous accueillir à bord de *Hackable n°10* des Éditions Diamond à destination de NFC city. Les issues de secours sont situées de chaque côté du magazine, à l'avant, au centre, à l'arrière. Veuillez attacher et ajuster votre ceinture de sécurité. Nous vous souhaitons un très bon et captivant voyage... PNC aux portes, armement des toboggans, vérification de la porte opposée.

Denis Bodor

Hackable Magazine
est édité par Les Éditions Diamond



10, Place de la Cathédrale - 68000 Colmar
Tél. : 03 67 10 00 20 – Fax : 03 67 10 00 21
E-mail : lecteurs@hackable.fr
Service commercial : cial@ed-diamond.com
Sites : www.ed-diamond.com
Directeur de publication : Arnaud Metzler
Rédacteur en chef : Denis Bodor
Réalisation graphique : Kathrin Scali
Responsable publicité : Valérie Fréchard,
Tél. : 03 67 10 00 27 v.frechard@ed-diamond.com
Service abonnement : Tél. : 03 67 10 00 20
Impression : pva, Landau, Allemagne
Distribution France : (uniquement pour les dépositaires de presse)
MLP Réassort : Plate-forme de Saint-Barthélemy-d'Anjou. Tél. : 02 41 27 53 12
Plate-forme de Saint-Quentin-Fallavier.
Tél. : 04 74 82 63 04
Service des ventes : Abomarque : 09 53 15 21 77
IMPRIMÉ en Allemagne - PRINTED in Germany
Dépôt légal : À parution,
N° ISSN : 2427-4631
Commission paritaire : K92470
Périodicité : bimestriel
Prix de vente : 7,90 €



Suivez-nous sur Twitter
@hackablemag

À PROPOS DE HACKABLE...

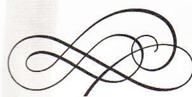
HACKS, HACKERS & HACKABLE

Ce magazine ne traite pas de piratage. Un **hack** est une solution rapide et bricolée pour régler un problème, tantôt élégante, tantôt brouillonne, mais systématiquement créative. Les personnes utilisant ce type de techniques sont appelées **hackers**, quel que soit le domaine technologique. C'est un abus de langage médiatisé que de confondre « pirate informatique » et « hacker ». Le nom de ce magazine a été choisi pour refléter cette notion de **bidouillage créatif** sur la base d'un terme utilisé dans sa définition légitime, véritable et historique.

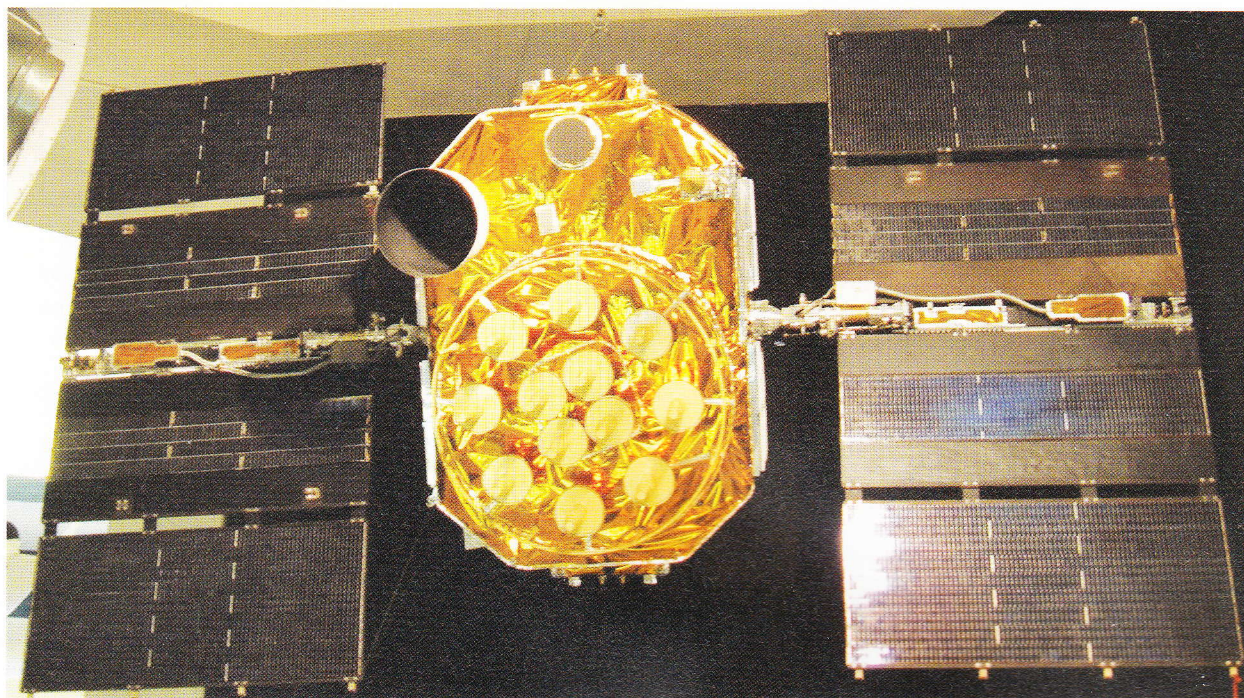
SOMMAIRE

GÉREZ UN RÉCEPTEUR GPS AVEC ARDUINO

Denis Bodor



Initialement, une technologie développée pour les forces armées et, dans une certaine mesure plus tard, les professionnels aisés, le GPS s'est très largement démocratisé ces dernières années. Après l'avoir vu se glisser dans nos voitures sous forme d'assistant, il est maintenant tout simplement partout puisqu'il est presque systématiquement présent dans n'importe quel smartphone. Nous pouvons également en faire usage avec une carte Arduino de façon relativement simple...



Ce satellite Boeing GPS-12 destiné au Block IIF est exposé au San Diego Aerospace Museum (USA). Il s'agit d'un exemplaire de remplacement au sol qui n'a jamais été lancé. D'autres satellites de remplacement sont en orbite pour pallier à un éventuel problème et garder la constellation fonctionnelle. (Crédit photo : Scott Ehardt / Wikipédia / Domaine Public)

L'utilisation du terme « GPS » est souvent un abus de langage auprès du grand public. GPS est l'acronyme de *Global Positioning System* et désigne une technologie permettant à un récepteur de connaître sa position (se géolocaliser), partout sur terre, en mer et dans le ciel (si les conditions sont favorables), en reposant sur les messages envoyés par quelques 30 satellites orbitant à une altitude de à 20200 km (71 ont été lancés, 33 sont normalement en service, mais 24 sont suffisants pour faire fonctionner l'ensemble du système).

Il est donc techniquement faux de dire qu'on « emporte son GPS », qu'on « a été mal guidé par son GPS » ou encore qu'on peut « localiser un véhicule grâce à son GPS ». L'objet auquel on fait ainsi référence est en réalité un équipement intégrant un récepteur GPS. De plus, la technologie GPS elle-même ne concerne que le positionnement dans l'espace sur trois axes (latitude, longitude et altitude). La planification d'un trajet ou le positionnement sur une carte n'est donc pas du ressort de la technologie GPS, pourtant malheureusement souvent incriminée à tort.

Enfin, il est important de noter que le GPS n'est pas le seul système disponible. GLONASS, géré par la Fédération de Russie, par exemple, propose le même type de service et connaît actuellement un succès grandissant. Il est d'ailleurs de plus en plus courant dans les équipements de positionnement pour randonneurs de voir GPS et GLONASS utilisés de concert. Plus en retrait, nous avons également QZSS pour *Quasi-Zenith Satellite System* (Japon) qui est complémentaire au GPS, BeiDou/COMPASS (Chine) qui devrait être mondialement utilisable en 2020, ou encore Galileo, le système européen qui est actuellement en test et devrait être également opérationnel en 2020 avec ses 30 satellites (ou pas).

Tous ces systèmes sont des GNSS ou *Global Navigation Satellite System*, des systèmes de positionnement par satellites. C'est un acronyme que vous retrouverez régulièrement en faisant des recherches sur le Web. Un autre est NMEA pour



Couramment appelé « GPS de randonnée », ce type de périphérique endurci (IPX7) intègre bien plus qu'un récepteur GPS+GLONASS. Il s'agit d'un système embarqué complet avec processeur, mémoire, affichage, lecteur microSD, système d'exploitation permettant bien plus que de simplement connaître sa position.

National Marine Electronics Association, suivi d'un numéro (NMEA 0183 par exemple) désignant un ensemble de spécifications pour la communication entre équipements maritimes (dont le GPS). Tous les récepteurs GNSS respectent normalement les spécifications NMEA 0183 (ou NMEA 2000) et communiquent donc avec le même protocole relativement simple sous la forme d'une liaison série et de « phrases » de texte en caractère ASCII. Enfin, vous verrez peut-être également le terme SBAS (*Satellite-Based Augmentation System*) ou *GNSS augmentation*. Il s'agit de systèmes de positionnement qui ne sont pas globaux, mais viennent en complément d'un GNSS comme GPS ou GLONASS pour ajouter de la précision, de la fiabilité ou de la disponibilité. Le QZSS japonais et ses trois futurs satellites (un seul actif pour l'instant) est un SBAS, non un GNSS. Et pour que l'image soit vraiment complète, il faut encore ajouter les systèmes comme WAAS (USA) ou EGNOS (EU) utilisant des stations terrestres et venant également en complément des GNSS pour augmenter la précision.

Je vous propose donc maintenant de découvrir une petite partie de cette technologie avec une simple carte Arduino et, bien entendu un récepteur GPS.

1. DIFFÉRENTS RÉCEPTEURS GPS

Il existe bien des types d'équipements embarquant un récepteur GPS. Ceux-ci peuvent être un smartphone, un système de navigation pour véhicule, un périphérique USB, un module électronique, un équipement pour hélicoptère multirotor (généralement appelé « drone civil »), un appareil endurci pour la randonnée, le VTT ou les activités sportives, etc.

Tous ces équipements ont en général un point commun qui se matérialise par une liaison série entre un récepteur GPS et le reste de l'électronique. Il est donc, en principe parfaitement possible d'utiliser n'importe quel type de matériel pour recevoir les informations GPS avec une carte Arduino. Bien entendu, le niveau d'intégration dans un smartphone ou un équipement de randonnée rend cette opération délicate, sans parler du coût et du risque de destruction.

Il reste donc trois solutions accessibles :

- Le module GPS qu'on peut trouver pour quelques 15€ ou 20€ sur des sites comme eBay : ils se présentent tous généralement de la même façon sous forme de modules comprenant un récepteur, un circuit d'alimentation et une antenne. Les récepteurs les plus courants dans cette catégorie sont sans le moindre doute ceux du fabricant ublox avec des modèles de la série

Neo-6 (GPS), Neo-7 (GPS + GLONASS) ou Neo-8 (GPS + GLONASS + BeiDou). Attention cependant, ces équipements et modules sont parfois interfacés uniquement en 3,3V alors que le choix devra naturellement se porter sur un circuit compatible 3V-5V.

- L'équipement pour contrôleur de vol de multirotors/drones. Dans une tranche de prix sensiblement plus haute que le matériel précédent, ces périphériques sont destinés à être intégrés dans des appareils volants. Ils sont donc généralement plus robustes qu'un simple module, car noyés dans la résine, et intègrent souvent un magnétomètre/compas/boussole en complément. Mais, d'autre part, ce type de matériel est presque toujours interfacé en 3,3V et nécessitera donc une adaptation des tensions lors d'une connexion à une carte Arduino (autre que Due ou Zero).
- Le périphérique USB GPS : ce type de périphérique est en voie de disparition, car la plupart des utilisateurs disposent maintenant d'une solution de géolocalisation sur leur smartphone. Fut un temps, un PC était nécessaire pour cela dès lors qu'on ne souhaitait pas opter pour un système intégré coûteux pour voiture par exemple. On retrouve aujourd'hui ces périphériques USB pour quelques 10€ ou 15€ sous la forme de fins de stock ou d'occasion (j'en ai même trouvé pour quelques euros sur un marché aux puces). C'est une solution économique utilisable, car le matériel se résume à un récepteur GPS série et une puce de conversion USB/série. Il suffit donc de démonter le tout, trouver les bonnes connexions et se débarrasser de la conversion série vers USB pour la remplacer par une carte Arduino. Il faut cependant bien comprendre que cela reste une loterie : certains circuits sont en 5V d'autres non, certains périphériques ont une interface dans un boîtier au milieu du câble, d'autres intègrent tout au niveau de l'antenne, et enfin, certains s'ouvrent en retirant quelques vis et d'autres sont thermocollés ou noyés dans la résine.

Voici notre « victime » : un récepteur GPS USB de marque obscure acquis pour quelques 3 malheureux euros sur un marché aux puces. L'appareil se compose d'un récepteur GPS, d'un module de conversion USB en plein milieu du câble et d'un connecteur USB.



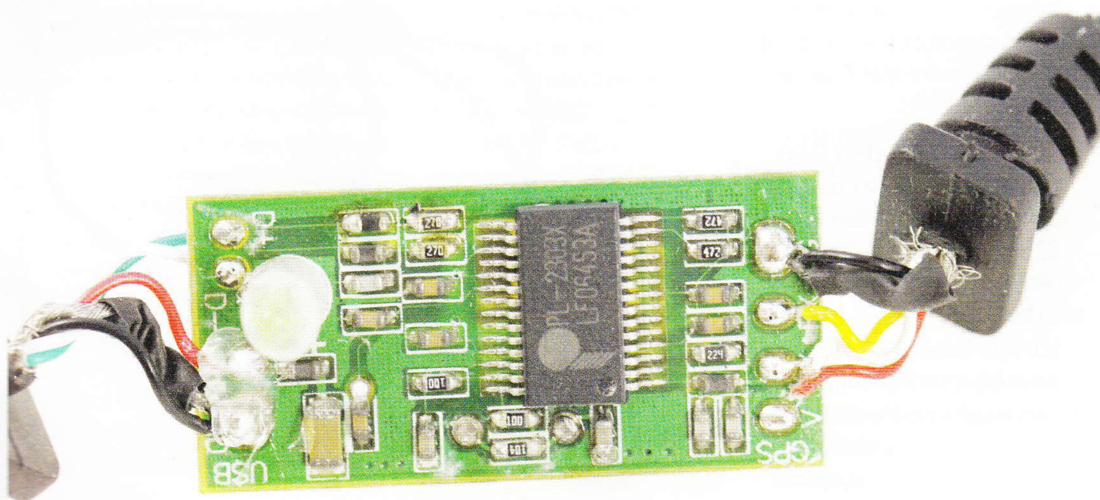
Dans tous les cas, après démontage, bricolage et éventuellement adaptation des tensions, on se retrouve avec un récepteur GPS envoyant ses données via une liaison série à 9600 bps ou 4800 bps, parfaitement utilisable avec une carte Arduino.

Les expérimentations du présent article ont été réalisées avec un récepteur GPS USB, obtenu pour 3€ sur un marché aux puces, duquel j'ai retiré le circuit de conversion USB/série (à base de Prolific PL-2303 qui pourra être réutilisé pour un autre projet).

La première étape dans cette démarche consiste à identifier le brochage côté récepteur. Une chance pour moi, le circuit imprimé était marqué de « gnd », « rx », « tx » et « vcc » au niveau des points de soudure. Si tel n'avait pas été le cas, la logique aurait été la suivante :

- Identifier en premier lieu la masse : ceci est relativement facile puisqu'il suffit de tester la continuité entre chaque connexion et une masse connue comme celle du connecteur USB ou de la puce USB/série dont on trouvera facilement la documentation en ligne.
- Une fois la masse trouvée, il faut repérer la broche l'alimentation (Vcc) : là encore, on peut s'aider de la documentation des composants présents et de leur brochage. On peut également parcourir les pistes depuis le côté USB puisque la position de Vcc sur une prise USB est connue et qu'il suffit alors de suivre tant bien que mal la liaison.
- Reste TX et RX, ou en d'autres termes l'envoi et la réception de données vers et depuis le récepteur GPS : on peut suivre les pistes depuis le convertisseur USB/série, brancher un oscilloscope, monter une led avec une résistance ou tout simplement essayer l'un puis l'autre (mais là, uniquement après être passé par l'étape suivante).

La seconde chose à faire pour arriver à ses fins est de s'assurer des tensions utilisées. Un simple multimètre en mesure de tension entre la masse et Vcc est suffisant. Je n'ai jamais, au grand jamais, vu de circuit de ce type fournissant une alimentation à une tension donnée tout en en utilisant une autre pour les signaux logiques. Il est donc totalement légitime de se dire que si la tension masse/Vcc est de 5 volts, les signaux sont aussi en 5 volts.



Le circuit situé en plein milieu du câble est construit autour d'un simple convertisseur USB/série (PL-2303). On a ici à gauche la connexion USB et à droite la liaison série avec le récepteur GPS.

Les broches sont repérées ? Les tensions mesurées ? Oui ? Alors, allons-y ! Il n'y a plus qu'à sortir la carte Arduino et voir ce que le récepteur raconte...

2. ARDUINO EN TANT QUE RELAIS : PARLE-MOI EN NMEA S'IL TE PLAÎT !

Notre premier croquis aura pour objet non pas d'obtenir des informations de positionnement directement intelligibles (par nous), mais de s'assurer que le récepteur fonctionne et transmette des informations utilisables.

En principe, ceci revient simplement à lire les données série à une certaine vitesse d'un côté et à les retransmettre à une autre vitesse sur le moniteur série Arduino. Pour ce faire, nous n'avons pas besoin d'une bibliothèque spécialisée en GPS et, avec certaines cartes Arduino, nous n'aurions pas besoin de bibliothèques du tout.

Comme il s'agit d'une étape de vérification, nous allons cependant faire reposer notre code sur trois bibliothèques dont une déjà intégrée à l'environnement (les deux autres étant disponibles via le gestionnaire de bibliothèques). Nous obtiendrons ainsi un squelette de croquis qu'il nous suffira, ensuite, de finaliser. Les bibliothèques en œuvre seront donc :

- *SoftwareSerial* : Celle-ci n'est utile que pour les cartes Arduino ne disposant que d'un seul port série matériel comme l'Arduino UNO par exemple. Nous utilisons ce port série pour le moniteur, mais il nous en faut également un pour connecter le récepteur GPS. Cette bibliothèque nous permet d'en ajouter un, totalement logiciel, qui s'utilisera comme son équivalent matériel (`Serial`). Avec des cartes comme l'Arduino Mega ou Leonardo en revanche, nous disposons de `Serial1` en plus de `Serial` et n'aurons donc pas besoin de cette bibliothèque.
- *TimerOne* : Il y a bien des manières de gérer les données arrivant de façon continue via une liaison série. Celle que nous choisirons ici consiste à régulièrement collecter les octets (caractères) arrivant et à les transmettre à la bibliothèque qui gérera le « dialecte » GPS (trames NMEA). Pour ce faire, nous utiliserons un *timer* et une interruption. Ceci nous coûte l'installation d'une bibliothèque et l'indisponibilité de quelques sorties analogiques (PWM), mais facilitera, en contrepartie l'écriture et la structuration du code. Concernant les *timers* et TimerOne, nous en avons détaillé le fonctionnement dans l'article consacré aux opérations atomiques.
- *Adafruit GPS* : Les récepteurs GPS respectent normalement les spécifications NMEA 0183 (ou NMEA 2000) qui décrivent, entre autres, la syntaxe utilisée pour transmettre les informations collectées. Il s'agit de données formatées en lignes de texte ASCII normalisées et conçues pour être lisibles par une machine et non un humain. Il est donc nécessaire d'interpréter ces lignes et d'en extraire les informations utiles. Voilà exactement le gros du travail à réaliser lorsqu'on utilise un récepteur GPS et précisément celui accompli par cette bibliothèque. Notez qu'il existe également MicroNMEA de Steve Marple, également disponible via le gestionnaire de bibliothèques. Celle-ci est plus complète, mais également sensiblement plus délicate à mettre en œuvre, et donc moins prompte à une découverte pédagogique du sujet en un nombre de pages limité (selon moi).

Une fois ces bibliothèques installées, il ne nous reste plus qu'à écrire le croquis :

```
#include <TimerOne.h>
#include <Adafruit_GPS.h>
#include <SoftwareSerial.h>

// RX sur 3, TX sur 2
SoftwareSerial softSerial(3, 2);

// objet pour représenter le récepteur
Adafruit_GPS GPS(&softSerial);

void setup(){
  // Moniteur série en 115200
  Serial.begin(115200);
  // Petit coucou
  Serial.println("Adafruit GPS test");
  // softserial via Adafruit_GPS en 4800
  GPS.begin(4800);

  // une interruption à chaque milliseconde
  Timer1.initialize(1000);
  // On appellera cette fonction
  Timer1.attachInterrupt(lectureGPS);

  delay(2000);
}

// fonction de lecture des données
void lectureGPS() {
  // l'objet relève simplement les informations
  char c = GPS.read();
  // Si un caractère est lu
  if(c)
    // on le copie sur le moniteur
    Serial.write(c);
}

// rien à faire ici tout se passe dans lectureGPS()
void loop() {
  // On ajoutera plus tard le code pour l'affichage
  // des données GPS
}
```

Les commentaires dans le code devraient être suffisants pour en comprendre le fonctionnement et la teneur. Nous nous contentons d'appeler `lectureGPS()` toutes les millisecondes afin de lire un caractère sur le port série logiciel via `GPS.read()` en 4800 bps (attention, la plupart des récepteurs utilisent la vitesse plus courante de 9600 bps). Cette fonction se charge de la lecture, mais également, en coulisse,

de l'accumulation des données et de la concaténation en messages/trames NMEA. Nous récupérons cependant l'éventuel caractère retourné dans la variable **c** et l'envoyons sur le moniteur série . Nous verrons donc apparaître dans celui-ci des choses comme :

```
Adafruit GPS test
00,129.86,251115,,,A*6F
$GPGGA,082630.000,4804.6301,N,00721.5518,E,1,08,1.1,209.7,M,47.9,M,,0000*5C
$GPGSA,A,3,16,01,18,08,27,22,11,21,,,,,1.6,1.1,1.2*37
$GPRMC,082630.000,A,4804.6301,N,00721.5518,E,0.00,129.86,251115,,,A*67
$GPGGA,082631.000,4804.6301,N,00721.5518,E,1,08,1.1,209.7,M,47.9,M,,0000*5D
$GPGSA,A,3,16,01,18,08,27,22,11,21,,,,,1.6,1.1,1.2*37
$GPRMC,082631.000,A,4804.6301,N,00721.5518,E,0.00,129.86,251115,,,A*66
$GPGGA,082632.000,4804.6301,N,00721.5518,E,1,08,1.1,209.7,M,47.9,M,,0000*5E
$GPGSA,A,3,16,01,18,08,27,22,11,21,,,,,1.6,1.1,1.2*37
$GPGSV,3,1,12,27,77,136,28,08,64,299,27,22,64,097,38,18,32,054,24*76
$GPGSV,3,2,12,11,23,276,20,16,21,187,17,01,13,260,17,32,13,201,23*77
$GPGSV,3,3,12,14,12,122,20,21,11,076,14,30,05,309,16,15,02,023,*74
```

Ceci est littéralement ce que transmet, de façon continue, le récepteur GPS. On trouve dans ces lignes les informations de positionnement, mais également la date et l'heure transmises par les satellites, les informations sur la qualité du signal et la fiabilité de la géolocalisation, la liste des satellites en vue et utilisés pour les calculs, etc.

La plupart de ces informations sont calculées par le récepteur d'après les données transmises par les satellites, mais pas toutes, certaines sont de simples traductions des signaux reçus. On peut donc dire que ce que vous voyez défiler sur votre écran est (presque) la transmission radio envoyée par des objets se trouvant à quelques 20000 km au-dessus de votre tête... dans l'espace !

Notez que les messages ou phrases NMEA débutent tous par **$** puis d'un premier mot suivi d'une liste d'éléments séparés par des virgules (la seconde ligne ici ne compte pas, elle est incomplète). Ces premiers mots, **GPGGA**, **GPRMC**, **GPGSV**... désignent le type de données présentes dans le reste de chaque phrase. Ces mots débutent tous ici par **GP**, mais ceci n'est pas systématique. Ces deux caractères forment un *talker ID* qui le plus souvent est **GP** pour GPS, mais selon le récepteur, vous pouvez aussi avoir **GN** pour GLONASS par exemple (la bibliothèque *Adafruit GPS* ne gère pas ces messages, mais MicroNMEA oui).

Deux solutions s'offrent alors à vous, soit apprendre ces spécifications par cœur pour lire « façon Matrix » les données, soit tout simplement laisser faire la bibliothèque *Adafruit GPS*. Comme tout programmeur qui se respecte, soyons feignants et laissons faire le sale boulot à un bout de code...

3. DÉCODONS TOUT ÇA !

Pour transformer notre croquis de démarrage en système plus intelligible, nous n'avons pas grand-chose à faire. En effet, la simple utilisation de **GPS.read()** suffit à « nourrir » la bibliothèque et nous n'avons qu'à afficher les données de façon régulière sur le moniteur série.

Mais avant cela, nous revenons simplement sur la fonction **lectureGPS()** en mettant en commentaire ou en supprimant les éléments qui peuvent interférer :

```
void lectureGPS() {
  char c = GPS.read();
}
```

Nous n'avons alors plus d'écho des trames NMEA sur le moniteur série. Nous en profitons également pour ajouter une fonction destinée à afficher proprement une valeur sur deux positions :

```
void print2digits(int val, Stream &console) {
  // si <10 alors on préfixe d'un 0
  if(val<0)
    console.print("0");
  console.print(val, DEC);
}
```

L'idée est ici d'afficher « 05 » si **val** est 5, « 42 » si la valeur est 42, etc. Cette fonction est destinée à être utilisée pour l'affichage de l'heure afin d'obtenir quelque chose comme « 09:12:08 » et non « 9:12:8 ». C'est un point « cosmétique », mais cela nous permet ici de voir qu'il est possible de passer en paramètre d'une fonction le flux représentant un port série. Cette fonction utilisée avec **print2digits(42, Serial)** pourra alors utiliser **console** comme **Serial**, et ce avec les mêmes méthodes, comme ici **print**. Pour changer le port utilisé, nous n'avons pas à changer notre fonction, mais juste ses arguments.

Nous pouvons alors nous lancer dans l'écriture de notre fonction **loop()** pour afficher plein de choses intéressantes :

```
#define PAUSE 2000
unsigned long previousMillis = 0;

void loop() {
  unsigned long currentMillis = millis();

  // On a un message NMEA
  if (GPS.newNMEAreceived()) {
    // S'il n'est pas lisible
    if (!GPS.parse(GPS.lastNMEA()))
      // on abandonne
      return;
  }

  // toutes les PAUSE millisecondes
  if(currentMillis - previousMillis >= PAUSE) {
    previousMillis = currentMillis;

    // on affiche les données de base
    Serial.print("\nHeure GMT: ");
    print2digits(GPS.hour,Serial); Serial.print(':');
    print2digits(GPS.minute,Serial); Serial.print(':');
    print2digits(GPS.seconds,Serial);
    Serial.println("");

    Serial.print("Date: ");
    Serial.print(GPS.day, DEC); Serial.print('/');
```

```
      Serial.print(GPS.month, DEC); Serial.print("/20");
      Serial.println(GPS.year, DEC);

      Serial.print("Fix: "); Serial.println((int)GPS.fix >0 ? "OUI" : "NON");

      // Si le récepteur GPS a pu calculer une position on
      // affiche plus d'informations
      if (GPS.fix) {
        Serial.print("Qualite: "); Serial.println(GPS.fixquality);
        Serial.print("HDOP: "); Serial.println(GPS.HDOP,2);
        Serial.print("Precision: "); Serial.print(GPS.HDOP*4,2); Serial.println("m");
        Serial.print("Position: ");
        Serial.print(GPS.latitudeDegrees, 4);
        Serial.print(", ");
        Serial.println(GPS.longitudeDegrees, 4);

        Serial.print("Vitesse: ");
        Serial.print(GPS.speed,2); Serial.print(" noeud(s)      ");
        Serial.print((GPS.speed*1.852),2); Serial.println(" km/h");
        Serial.print("Altitude: "); Serial.println(GPS.altitude);
        Serial.print("Nbr Satellites: "); Serial.println((int)GPS.satellites);
      }
    }
  }
```

La boucle dépend de l'utilisation de **millis()** permettant d'utiliser une temporisation sur l'affichage tout en exécutant le plus souvent possible **GPS.parse(GPS.lastNMEA()** pour provoquer l'analyse des trames NMEA, après nous être assuré qu'une nouvelle était disponible avec **GPS.newNMEAreceived()**. Cette gestion de la réception de trames est faite en coulisse, mais nous devons toutefois nous charger de provoquer l'analyse. Si **GPS.parse()** retourne une valeur supérieure à zéro, c'est signe que la trame en question n'est pas traitable et la boucle stoppe là et repart pour un tour.

Toutes les **PAUSE** millisecondes (définies à 2000 en début de croquis avec un **#define PAUSE 2000**), nous collectons les informations et affichons les données. Ceci se fait en deux temps, car la réception GPS ne découle pas forcément sur un positionnement (un *fix*). Pour ce faire, il nous faut au moins les messages de quatre satellites et le calcul de la solution.

Ce calcul n'est pas notre affaire, ni même celui de la bibliothèque, mais celle du récepteur lui-même. Celui-ci peut ainsi parfaitement obtenir une date et une heure (GMT) et ne pas pouvoir nous fournir un positionnement fiable. Nous tentons donc systématiquement d'afficher

GPStest est une application Android gratuite fournissant énormément d'informations sur le positionnement GPS. Celle-ci pourra vous être très utile afin de vérifier et confirmer les données obtenues avec votre montage Arduino.



les données temporelles, mais vérifions préalablement avec la valeur booléenne (vrai/faux) contenue dans **GPS.fix** si nous avons des informations de géolocalisation à afficher.

Si cette condition est vérifiée, il s'agit tout au plus d'envier du texte et le contenu de variables au moniteur série à grands coups de **print** et de **println**. Parmi ces informations, certaines nécessitent un minimum d'explications :

- **GPS.fixquality** n'est pas une valeur numérique indiquant la qualité proportionnelle du positionnement. Au sens NMEA du terme, il s'agit davantage de signifier du type de qualité, avec des choses comme 0 pour « invalide », 1 pour GPS civil (SPS), 2 pour GPS différentiel ou DGPS, etc. Avec un simple récepteur GPS, si une solution peut être calculée, vous verrez certainement toujours 1 ici.

- **GPS.HDOP** est une valeur indiquant un facteur de précision inverse. HDOP signifie *Horizontal Dilution Of Precision* et, sans entrer dans le détail, plus cette valeur est proche de 1, plus le niveau de précision est important. Pour obtenir une précision approximative en mètres, on utilise cavalièrement la valeur HDOP multipliée par la résolution maximum du GPS civil (4 mètres). Ceci peut paraître surprenant, car les constructeurs annoncent tantôt des précisions allant jusqu'au mètre, mais ceci est généralement le fruit de l'utilisation de services ou de calculs complémentaires. Dans les faits, le GPS SPS c'est 4 mètres.

- **GPS.speed** contient la vitesse qui, pour ce genre de montage sédentaire, sera certainement à zéro. Attention, cette valeur est exprimée en nœud et non en kilomètres par heure. Nous devons donc nous plier d'une petite multiplication pour la conversion.

- **GPS.satellites** enfin, contient le nombre de satellites utilisés pour le calcul. Quatre sont suffisants pour un positionnement, mais la précision sera bien meilleure avec davantage de satellites.

La bibliothèque Adafruit GPS n'utilise pas tous les types de trames NMEA, mais uniquement :

- **GPGGA** (*Global Positioning System Fix Data*) : date, heure, longitude, latitude, qualité de positionnement, nombre de satellites, HDOP, altitude ;

- et **GPRMC** (*Recommended minimum specific GPS/Transit data*) : date, heure, longitude, latitude, vitesse.

Je vous recommande fortement la lecture du code de la méthode **Adafruit_GPS::parse()** dans le fichier **Adafruit_GPS.cpp**, s'occupant de l'analyse des trames NMEA et permettant de peupler les variables. C'est très instructif et, pourquoi pas, vous pourrez y apporter des modifications afin d'obtenir davantage d'informations (comme la liste des satellites utilisés, identifiés par leur PRN).

Notre nouvelle version du croquis (disponible en version intégrale sur le dépôt GitHub du magazine), une fois chargée dans la carte Arduino, nous affichera alors des lignes comme :

```
Heure GMT: 16:41:19
Date: 25/11/2015
Fix: 1
Qualite: 1
HDOP: 1.10
Precision: 4.40m
Position: 48.0771, 7.3591
Vitesse: 0.00 noeud(s)     0.00 km/h
Altitude: 176.90
Nbr Satellites: 7
```

Nous retrouvons ici tout ce qu'il nous faut. Notez au passage que la position est affichée en degrés et sera directement utilisable avec un service comme Google Maps ou OpenStreetMap.

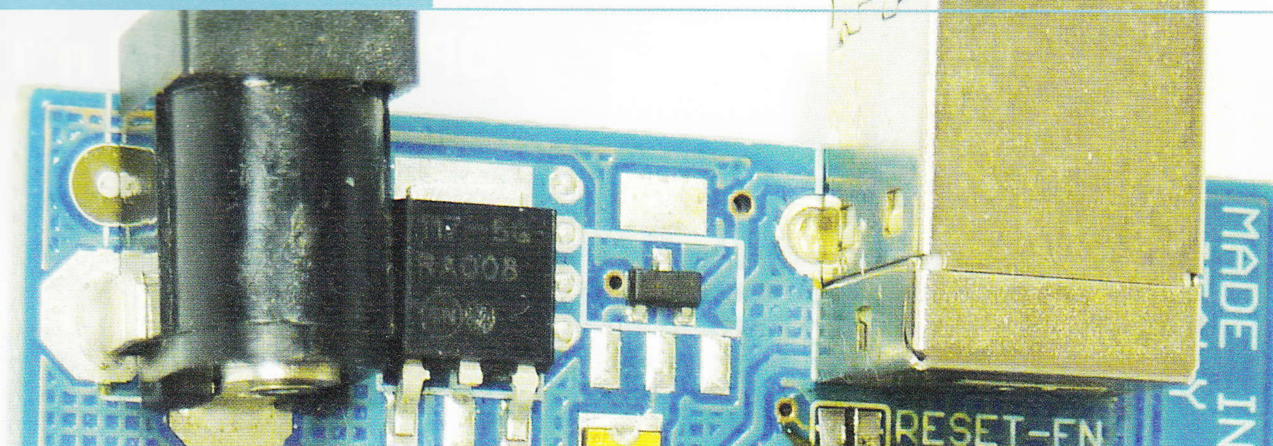
4. POUR FINIR

Stoppons là, puisqu'il faut bien terminer l'article à un moment. Nous avons ici mis en œuvre un récepteur GPS de récupération sans trop de difficultés et surtout avons appris énormément de choses. C'est d'ailleurs là l'aspect le plus intéressant, car à partir de cette première expérience, il devient possible d'envisager bien des applications. Dès lors qu'un montage est en mesure de savoir où il se trouve, des choses intéressantes s'ouvrent à nous : créer un périphérique de positionnement autonome avec un écran LCD HD44780 ou même un afficheur graphique, commencer la construction d'un robot autonome (*rover* d'exploration) ou encore simplement se servir du récepteur comme source d'horloge...

Mais il est possible de partir sur une voie bien plus intéressante encore en étudiant les spécifications NMEA (**http://aprs.gids.nl/nmea/** regroupe tout cela de façon très intelligible) et en améliorant la bibliothèque Adafruit ou en développant la vôtre. Dans la vingtaine de types de trames que votre récepteur peut vous envoyer, on trouve par exemple les identifiants des satellites utilisés, mais aussi... leur position dans le ciel (trame `GPGSV`) !

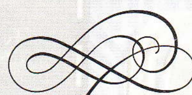
Même si vous n'avez pas besoin de géolocalisation, cette simple information à quelque chose de magique et l'idée d'un montage « pointant du doigt » ces condensés de technologie placés dans l'espace que sont les satellites artificiels, est tout à fait séduisante que ce soit avec des servomoteurs, des leds disposées sur un dôme ou, pourquoi pas, dispersées au plafond, au-dessus de votre lit...

Et si vous êtes amateur d'astronomie, pourquoi ne pas tenter d'utiliser ces informations et quelques moteurs, pour pointer automatiquement votre télescope ? Les idées ne manquent pas (mais la place dans ce magazine oui). DB

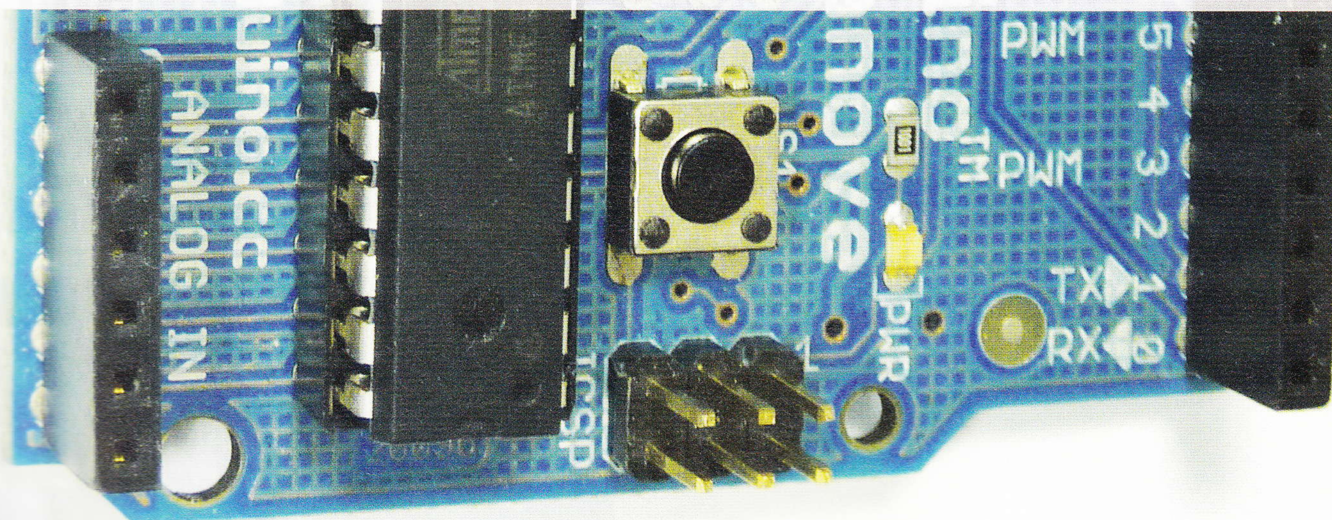


COMMUNICATION PAR LUMIÈRE VISIBLE SUR ARDUINO

Jonathan Piat (jonathan.piat@iut-tlse3.fr)
Enseignant-chercheur à l'IUT GEII de l'Université Paul Sabatier, Toulouse



Visible Light Communication est un moyen de communication utilisant la portion visible du spectre lumineux. Cette technologie émergente vise à fournir une alternative aux transmissions radio en utilisant l'éclairage ambiant. Dans cet article, nous mettrons en œuvre cette technologie sur plateforme Arduino.



1. VISIBLE LIGHT COMMUNICATION OU COMMUNICATION PAR LUMIÈRE VISIBLE

Visible Light Communication (VLC) est un moyen de communication utilisant la portion visible du spectre lumineux. L'objectif de cette technologie est d'utiliser des moyens d'éclairage existants (plafonniers, indicateurs...) et les valoriser pour la transmission d'informations. Le principe de cette technologie est apparu en 1880, mais les applications grand public commencent seulement à émerger.

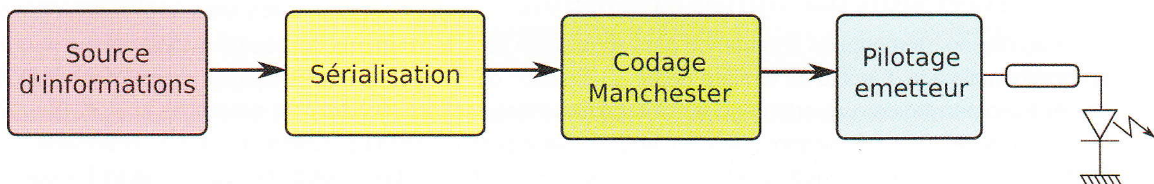
Les capacités de transmission s'avèrent importantes (débit supérieur au Gbit/s) et présentent de multiples avantages face aux autres technologies sans fil. En effet, contrairement au Wi-Fi, il n'y a pas de risque pour la santé, la consommation électrique est presque nulle (si on considère que la transmission de données n'est pas le but primaire de l'éclairage) et l'information s'arrête avec la lumière tandis que les autres technologies sans fil rayonnent au-delà de la portée souhaitée (problème de sécurité).

La transmission de l'information se fait par modulation de la lumière (à des fréquences imperceptibles) en utilisant une ou plusieurs LED. Le récepteur peut soit être constitué d'un détecteur spécialisé ou utiliser – lui aussi – une ou plusieurs LED afin de pouvoir, avec le même composant, jouer le rôle d'émetteur ou de récepteur (effectivement, les LED peuvent servir à détecter la lumière).

Par la suite, nous allons développer une chaîne de transmission complète basée sur les technologies VLC. Le code complet de cet exemple peut être téléchargé sur **https://github.com/jpiat/arduino**.

2. PRINCIPES DE FONCTIONNEMENT D'UNE CHAÎNE D'ÉMISSION SIMPLIFIÉE

Le schéma d'émission proposé est le suivant :



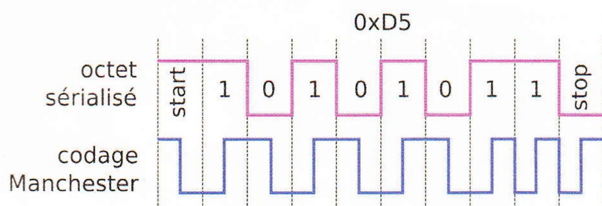
2.1 Sérialisation d'un octet

La sérialisation consiste à prendre l'information sous sa forme octet (8 bits de données) pour générer une séquence de 8 fois un bit. Pour des soucis de récupération de la synchronisation, nous ajoutons à cette séquence de 8 bits, deux bits, soit *start* au début de la séquence et *stop* à la fin de la séquence. Ces deux bits permettront au récepteur de se synchroniser pour la réception d'un octet.

2.2 Codage Manchester

Le codage Manchester transforme un bit de données en une séquence de deux symboles. Après le codage Manchester, un bit sera non plus représenté par un niveau logique '1' ou '0', mais par une séquence "01" pour un bit à '0' et une séquence "10" pour un bit à '1'. Ce codage possède plusieurs propriétés intéressantes pour les télécommunications :

- La récupération de la synchronisation (horloge permettant d'analyser les symboles de la transmission) est facilitée par les transitions du signal.



- L'énergie moyenne de la transmission est constante. En effet, en admettant la transmission d'une longue séquence de symboles, le temps ou le signal est à l'état logique '1' et est à peu près égal au temps où le signal est à l'état logique '0'.

Cette deuxième propriété est très intéressante dans notre cas, puisque cela signifie que la quantité de lumière émise sera identique, quelle que soit l'information envoyée.

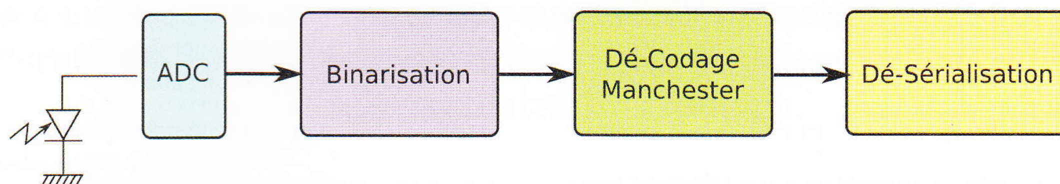
2.3 Pilotage de la LED

L'émetteur utilisé pour la transmission est une simple LED associée à une résistance de limitation de courant. Le courant délivrable par chaque entrée/sortie d'un Arduino étant de 25 mA, on choisira la valeur de la résistance selon la couleur de la LED, rouge ou verte pour ~120 ohms, bleu pour ~70 ohms. La LED utilisée devra être en boîtier cristal (non diffusé). Pour obtenir un éclairage blanc, on peut associer les trois couleurs (LED RGB).

2.4 Vitesse d'émission

Le débit symbole de notre communication (débit du code Manchester) sera limité à 1200 bauds. On doit pouvoir obtenir des vitesses plus grandes avec ce montage, mais l'expérience montre que ce débit permet une communication stable. Ce débit symbole permet donc un débit bit de 600 bits/s et un débit octet de 60 octets par seconde (chaque octet utilisant 10 bits dans notre codage avec *start* et *stop*).

3. PRINCIPES DE FONCTIONNEMENT D'UNE CHAÎNE DE RÉCEPTION SIMPLIFIÉE



La chaîne de réception visant à produire la donnée émise (ce qui paraît logique), les étapes de décodage du signal s'enchaînent dans l'ordre inverse de la chaîne d'émission.

3.1 Conversion du signal lumineux

Cette chaîne de réception commence par l'acquisition de la tension aux bornes de la LED qui est proportionnelle à la quantité de lumière reçue. L'utilisation d'un convertisseur analogique numérique (ADC) de l'Arduino permet de convertir cette tension en une valeur entière (échantillon) qui pourra ensuite être traitée. Lors de la communication, la quantité de lumière perçue par la LED varie au rythme du codage Manchester généré par l'émetteur. Une résistance de ~1 Mohms est mise en parallèle avec la LED pour limiter la capture par l'ADC du bruit électromagnétique ambiant (en limitant l'impédance d'entrée de l'ADC).

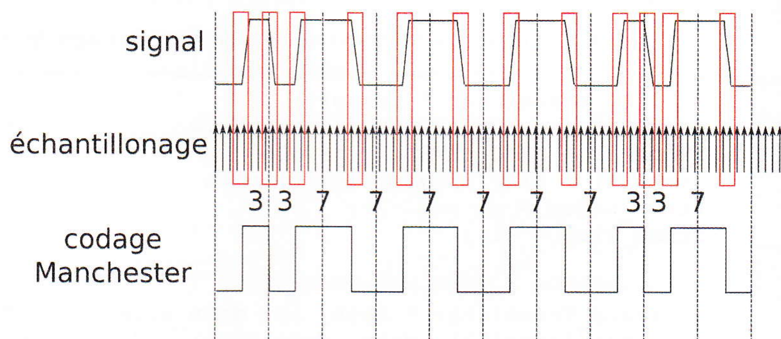
3.2 Décodage Manchester

Afin de pouvoir récupérer les symboles du code Manchester, il faut dans un premier temps discriminer les '1' et les '0' (on parle de « binarisation »). Pour ce faire, il existe deux possibilités :

- déterminer un seuil sur les échantillons qui permet de discriminer les '1' et les '0' ;
- utiliser les propriétés du code Manchester pour détecter les transitions au lieu des niveaux.

La deuxième méthode est la plus fiable dans le cas du codage Manchester et permet de s'affranchir du niveau de bruit ambiant plus facilement. La méthode utilisée ici commence par la détection d'un front sur le signal (échantillon courant supérieur ou inférieur à l'échantillon précédent). Lorsqu'un front est détecté, un '1' ou un '0' sont stockés dans un mot binaire pour un front montant et respectivement un front descendant. Une fois le front détecté, on compte le nombre d'échantillons stables depuis le front jusqu'au prochain front ce qui permettra de déterminer le nombre de '1' ou de '0' à stocker dans le mot binaire.

Une fois les symboles du code Manchester reçus, il reste à les interpréter puis à désérialiser la donnée. En fait, ce n'est pas si simple, car le codage Manchester, malgré toutes ses propriétés, pose un problème de détection de la phase. Dans le cas où l'émetteur envoie la séquence "11" qui, codée en Manchester, donnera "1010", on ne peut garantir à quel moment le récepteur va commencer à échantillonner la donnée (synchronisation). Si celui-ci commence à lire les symboles à partir du deuxième symbole, il percevra la séquence "010" qui sera alors interprétée comme un '0' suivi de quelque chose d'encore non déterminé. Il faut donc s'assurer de la mise en phase de l'émetteur et du récepteur. Cette mise en phase ne peut se faire que par l'intermédiaire d'un protocole de communication contenant des symboles garantissant la mise en phase. Nous définirons ce protocole dans la section 4.



3.3 Désérialisation

La désérialisation consiste en la détection du bit de *start* et du bit de *stop* dans le code Manchester. Ces deux symboles sont séparés de 16 symboles Manchester codant pour les 8 bits de l'octet à récupérer. Une bonne désérialisation ne se contente pas de détecter *start* et *stop* séparés de 16 symboles, mais plutôt de détecter une séquence stop | start | 16 symboles | stop. Une fois le cadrage *start stop* validé, il suffit de décoder les 16 symboles Manchester en 8 bits de données.

4. PROTOCOLE DE COMMUNICATION

L'étape 3.2 du récepteur a permis de révéler un problème de synchronisation. Nous définissons donc un protocole permettant de s'en affranchir. Ce protocole de couche physique consiste en une trame de données comportant des octets de synchronisation (préambule) et des octets de signalisation (début et fin des données).

Le format de trame adopté est le suivant :

- le préambule est formé de 3 octets avec la valeur 0xAA et un octet 0xD5 qui « casse » la monotonie ;
- l'octet STX (0x02) indique le début des données ;
- les données sont d'une taille maximum de 32 octets ;
- l'octet ETX (0x03) indique la fin de la transmission.

préambule			STX	données			ETX
0xAA	0xAA	0xAA	0x02	...	...	...	0x03

Le préambule permet à l'émetteur et au récepteur de se synchroniser. La suite d'octet 0xAA pourrait permettre de récupérer une synchronisation fine, et la donnée 0xD5 permet au récepteur de déterminer la phase de synchronisation. La charge utile est volontairement limitée en taille (32 octets) afin d'assurer que la synchronisation sera valide durant toute la durée de la transmission.

5. CODAGE DE L'ÉMETTEUR

Le programme côté émetteur consiste en une fonction de mise en trame des données à envoyer.

```
void init_frame(char * frame){
  memset(frame, 0xAA, 3); // préambule
  frame[3] = SYNC_SYMBOL ; //0xD5
  frame[4] = STX; //début des données
  frame_index = -1 ;
  frame_size = -1 ;
}

int create_frame(char * data, int data_size, char * frame){
  memcpy(&(frame[5]), data, data_size); //copie des données dans la trame
  frame[5+data_size] = ETX; //positionnement du caractère fin des données
  return 1 ;
}

int write(char * data, int data_size){
  if(frame_index >= 0) return -1 ; // une trame est en cours d'émission
   if(data_size > 32) return -1 ; // la taille des données est trop grande
  create_frame(data, data_size,frame_buffer);
  ATOMIC_BLOCK(ATOMIC_RESTORESTATE){ //protection contre les accès concurrents
    frame_index = 0 ; // une trame est à transmettre
    frame_size = data_size + 6 ; // initialisation de la taille de la trame
  }
  return 0 ;
}
```

On définit également une fonction appelée périodiquement pour l'émission des symboles Manchester codant la trame.

```
void to_manchester(unsigned char data, unsigned long int * data_manchester){
  unsigned int i ;
 (*data_manchester) = 0x02 ; // symbole STOP
 (*data_manchester) = (*data_manchester) << 2 ;
  for(i = 0 ; i < 8; i ++){
    if(data & 0x80) (*data_manchester) |=  0x02  ;
    // données en commençant par les poids faibles
    else (*data_manchester) |= 0x01 ;
    (*data_manchester) = (*data_manchester) << 2 ;
    data = data << 1 ; // to next bit
  }
  (*data_manchester) |= 0x01 ; //symbole START
}
```

```
//fonction périodique de l'émetteur
void emit_half_bit(){
     if(manchester_data & 0x01){ // envoie un symbole du code manchester
       SET_LED();
     }else{
       CLR_LED();
     }
     bit_counter -- ;
     manchester_data = (manchester_data >> 1);
     if(bit_counter == 0){
        manchester_data = 0xAAAAAAAA ;
        // si rien à envoyer on envoie des '1' codes en Manchester
        if(frame_index >= 0 ){
          if(frame_index < frame_size){
            to_manchester(frame_buffer[frame_index], &manchester_data);
            //conversion en Manchester
            frame_index ++ ;
          }else{
            frame_index = -1 ;
            frame_size = -1 ;
          }
        }
        bit_counter = WORD_LENGTH * 2 ; // 20 nouveaux bits à envoyer
     }
}
```

Cette dernière fonction est appelée 1200 fois par seconde pour générer les symboles Manchester à partir de la trame de données. La périodicité de l'appel est assurée par des interruptions générées par le timer 1 de l'ATmega328p. Le timer est configuré à l'aide de la bibliothèque TimerOne qui permet de régler la périodicité du timer et d'associer une fonction d'interruption au timer. L'accès à la sortie pilotant la LED ne se fait pas en utilisant la fonction usuelle `digitalWrite()`, mais par l'intermédiaire d'une macro qui permet un changement plus rapide de l'état de la sortie.

```
void setup() {
  OUT_LED(); // initialisation en sortie de la GPIO contrôlant la LED
  init_frame(frame_buffer); // initialisation de la trame de données
  Timer1.initialize(833); //initialisation du timer pour une fréquence de 1200Hz
  Timer1.attachInterrupt(emit_half_bit);  //configuration de la fonction périodique
}
```

La fonction de test de l'émetteur envoie en continu la donnée « Hello World! ».

```
void loop() {
  if(write("Hello World", 11) < 0){
  // la donnée ne peut être envoyée, car émetteur occupé
    delay(10);
  }
}
```

6. CODAGE DU RÉCEPTEUR

Le codage du récepteur est un peu moins aisé que dans le cas de l'émetteur. En effet, au niveau du récepteur il faut, tout d'abord, convertir les valeurs analogiques lues sur le convertisseur de l'Arduino en une séquence binaire pour commencer le décodage Manchester et la désérialisation.

Comme évoqué précédemment, le décodage des symboles Manchester se fera par détection des fronts. Le code de détection des fronts est exécuté à quatre fois la vitesse de transmission des symboles. La fonction **analogRead()** du langage Arduino n'est pas utilisée. En effet, cette fonction déclenche une conversion puis attend le résultat, ce qui ralentit grandement le système. Les fonctions utilisées ici permettent de déclencher une conversion et de lire le résultat. Ce fonctionnement permet de lire le résultat de la conversion précédente puis de déclencher une conversion pour l'itération suivante, et ainsi limiter le temps d'attente de conversion.

```
void sample_signal_edge(){
  char edge_val ;
  int sensorValue  = ADC_read_conversion(); //lecture du convertisseur
  ADC_start_conversion(SENSOR_PIN);
  // déclenchement d'une conversion pour la prochaine itération
  if((sensorValue - oldValue) > EDGE_THRESHOLD) edge_val = 1 ; // front montant
  else if((oldValue - sensorValue) > EDGE_THRESHOLD) edge_val = -1;
  // front descendant
  else edge_val = 0 ; //signal stable
  oldValue = sensorValue ;
  if(edge_val == 0){
    if( steady_count < (4 * SAMPLE_PER_SYMBOL)){
    // incrémentation du compteur de période
      steady_count ++ ;
    }
  }else{
          new_word = insert_edge(&shift_reg, edge_val, steady_count, &(dist_last_
sync), &detected_word);  //insertion du front dans le code Manchester
          if(dist_last_sync > (8 * SAMPLE_PER_SYMBOL)){ // limitation de la période
            dist_last_sync = 32 ;
          }
          steady_count = 0 ;
          // attente d'un prochain front, mise à 0 du compteur de période
        }
}
```

La fonction **insert_edge()** sert à ajouter le front dans le mot en code Manchester. Cette fonction retourne 1 si le front ajouté a permis le traitement d'un octet (détection du cadrage start/stop). Une fois un octet détecté, celui-ci est traité dans la boucle principale du programme.

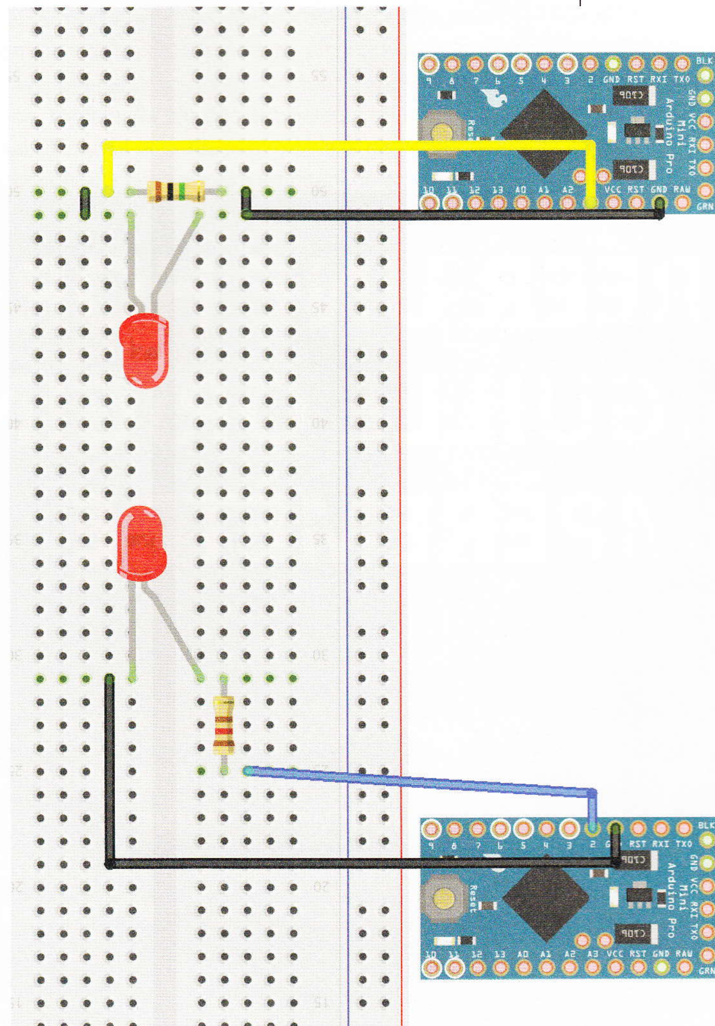
```
void loop() {
  int i;
  unsigned char received_data;
  char received_data_print ;
  int nb_shift ;
  int byte_added = 0 ;
  if(new_word == 1){ // un octet a été reçu
    received_data = 0 ;
    for(i = 0 ; i < 16 ; i = i + 2){ //décodage Manchester
             received_data = received_data << 1 ;
             if(((detected_word >> i) & 0x03) == 0x01){
                 received_data |= 0x01 ;
             }else{
                 received_data &= ~0x01 ;
             }
    }
```

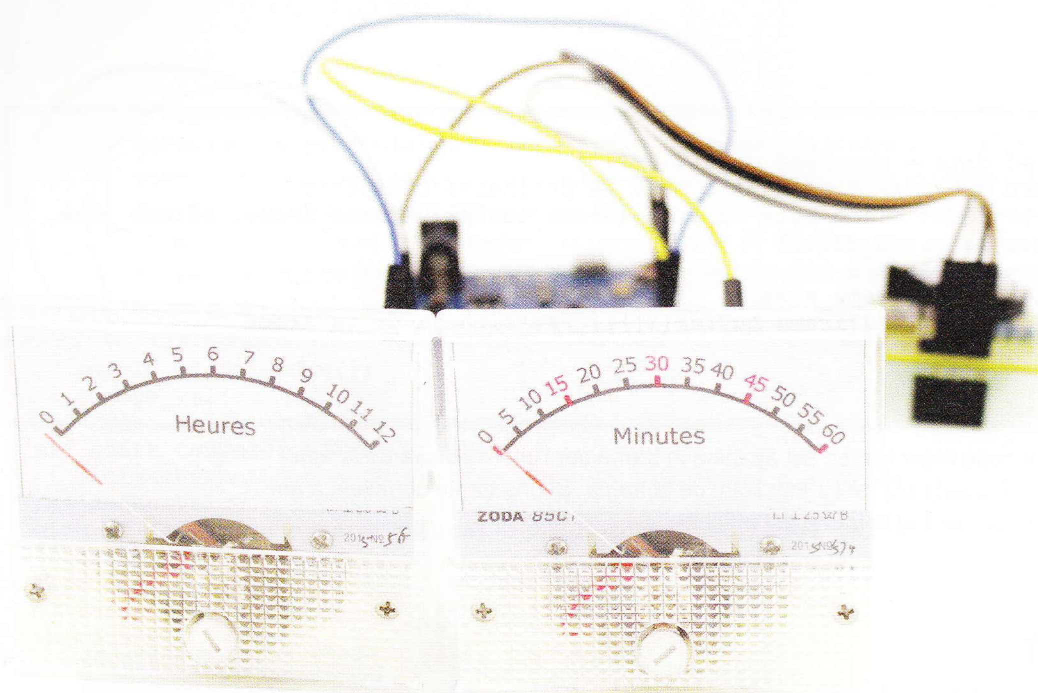
```
    received_data = received_data & 0xFF ;
    new_word = 0 ; // mise à zéro pour la prochaine réception
    if((byte_added = add_byte_to_frame(frame_buffer, &frame_index, &frame_size,
&frame_state,received_data)) > 0){
 //une trame complète a été reçue !
      frame_buffer[frame_size] = '\0';
      Serial.println(&(frame_buffer[1])); //affichage de la trame
    }
  }
}
```

L'octet extrait du code Manchester, est ajouté à la trame par l'intermédiaire de la fonction **add_byte_to_frame()** qui met à jour l'état de la trame. Si une trame complète a été reçue, la fonction retourne 1 et la trame est affichée.

7. TEST ET ÉVALUATION

Ce code a été testé à l'aide de deux cartes Arduino Pro-mini, l'une jouant le rôle d'émetteur et l'autre de récepteur. Une LED rouge haute luminosité est utilisée aussi bien du côté de l'émetteur que du récepteur. Les tests montrent que la vitesse de communication peut aller au-delà de 600 bit/s avec des distances ~10 cm. D'autres tests avec d'autres LED ont montré que la distance de communication peut atteindre le mètre selon la focalisation de la LED et l'alignement de l'émetteur et du récepteur. Des tests ont aussi été effectués avec des diodes RGB (rouge/vert/bleu) pour mixer les couleurs et obtenir une lumière blanche. Ces LED permettent la communication, mais toutes les couleurs ne se valent pas (le rouge offre le meilleur rapport vitesse/distance semble-t-il). D'autres tests sont en cours pour utiliser une LED RGB 1W et ainsi intégrer la transmission de données dans un véritable éclairage. JP





CRÉER UNE HORLOGE À AIGUILLE ORIGINALE : AMPÈREMÈTRE

Denis Bodor

Il existe bien des façons de créer une horloge, mais aussi bien des façons d'afficher votre appréciation de l'électronique. Pourquoi ne pas combiner les deux et utiliser un élément très représentatif du domaine pour vous composer une horloge à base d'ampèremètres analogiques et étonner vos ami(e)s avec de sympathiques cadrans à aiguille presque vintages ?


NIVEAU



TEMPS
10 minutes


BUDGET
30 €

Tout le monde a besoin de connaître l'heure et même si aujourd'hui il est très courant de simplement la lire sur son smartphone en même temps que d'éventuelles notifications, les montres et horloges sont plus que jamais des objets utiles, intéressants, beaux et prompts à véhiculer le style de leur propriétaire. Notre style à nous c'est l'électronique et il y a bien des manières de l'exprimer avec une horloge (oui, nous parlons ici d'un objet à poser sur un bureau, non à s'attacher au poignet). Horloges binaires, à lampes, pleines de fils, avec un côté vintage, intégrées dans un bureau, affichant en transparence à travers du bois de placage, composées à partir d'un matériel de récupération quelconque... les idées ne manquent pas.

Ce qui nous intéressera ici cependant repose sur un élément qu'on ne trouve plus beaucoup dans nos équipements de laboratoire ou d'atelier : un cadran à aiguille ou plus exactement un ampèremètre analogique magnéto-électrique (rien que ça !).

LE PRINCIPE

Un ampèremètre analogique, du type cité précédemment, n'est pas nécessairement un appareil de mesure en soi. Également appelé galvanomètre, il s'agit simplement d'une bobine mobile montée autour d'un aimant permanent ou entre deux aimants, le tout retenu par un ressort permettant de ramener la partie mobile en son point d'origine. Une aiguille est montée sur la partie mobile et permet d'amplifier le mouvement et lire la valeur à mesurer, directement sur une échelle graduée.

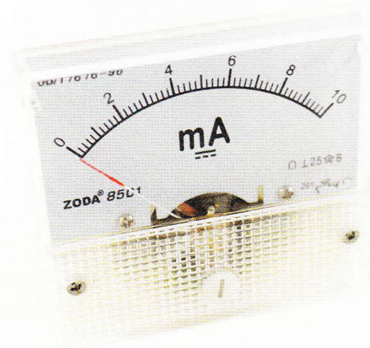
Lorsqu'un courant traverse la bobine, un champ magnétique est créé dans cette dernière qui s'aligne alors sur les pôles de l'aimant fixe tout en étant retenu par le ressort. L'ensemble du montage est calibré et conçu pour placer l'aiguille à la position correspondant à la valeur du courant qui circule. Lorsque le courant est coupé, la bobine, et donc l'aiguille, retournent à la position zéro par l'action du ressort.

Cette invention est le fruit de l'imagination de Luigi Galvani (1737-1798) et d'autres, parmi lesquels André-Marie Ampère (1775-1836) contribua à son perfectionnement.

Pendant longtemps, l'ampèremètre analogique était utilisé comme instrument de mesure, ainsi que le voltmètre analogique. Les deux équipements sont à présent en voie de disparition, remplacés par des équivalents numériques, plus solides, plus efficaces et perturbant moins le circuit sur lequel ils sont connectés. Il est d'ailleurs amusant de remarquer que le voltmètre analogique est un ampèremètre analogique affublé d'une résistance en série, alors que l'ampèremètre numérique est un voltmètre avec une résistance de shunt. C'est la magie de la loi d'Ohm, courant et tension sont intimement liés (U=R*I).

Mais les ampèremètres analogiques, sous la forme d'un simple cadran, sont toujours fabriqués et vendus. Ils trouvent leur place dans bien des réalisations, en particulier en électronique analogique, car leur fonctionnement simple est souvent plus économique qu'un système d'affichage numérique nécessitant généralement l'utilisation d'un microcontrôleur.

L'ampèremètre analogique est donc initialement un dispositif de mesure d'un courant circulant dans un montage, mais il est possible de voir cela de façon totalement différente : un dispositif magnéto-électro-mécanique qu'il est possible de contrôler en ajustant simplement un courant. Plus on fera passer de courant dans l'ampèremètre, plus on fera bouger l'aiguille qui donc nous permettra d'indiquer ce qui nous chante : direction, niveau, taux, température, vitesse ou... l'heure bien sûr !

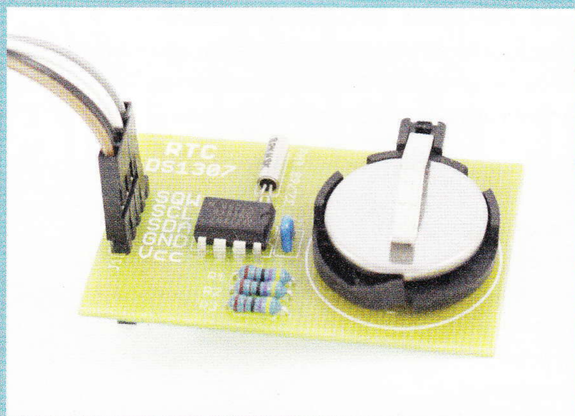


Un ampèremètre analogique comme celui-ci est en réalité un simple galvanomètre. Ici, ce modèle permet de mesurer un courant entre 0 et 10 mA, il sera cependant préférable d'opter pour une version 0-1 mA qu'on pourra alors utiliser en quantité plus importante.

CE QU'IL VOUS FAUT

- Une carte Arduino : techniquement, n'importe quelle carte fera l'affaire, Arduino ou autre. Notez cependant que nos calculs ont été faits sur la base de sorties en 5 volts. Si vous souhaitez utiliser une carte comme l'Arduino Due ou une Ti Launchpad MSP430, toutes deux en 3,3 volts, il faudra refaire les calculs et utiliser des résistances d'une valeur adaptée.

- Un module RTC DS1307 : c'est un composant courant comprenant généralement le circuit intégré DS1307, un quartz, quelques composants passifs, un emplacement pour pile bouton type CR2032 et tantôt la pile elle-même. On trouve ce type de module un peu partout sur le Web et bien entendu sur eBay pour un prix allant de 1 à 5 euros. Notez que nous partons ici du principe que nous travaillerons en 5V et de ce fait que le DS1307 sera parfaitement adapté. Pour une carte en 3,3 volts, il faudra opter pour un module DS1338 par exemple, pouvant être utilisé avec des tensions de 3V à 5V, mais sensiblement plus cher.



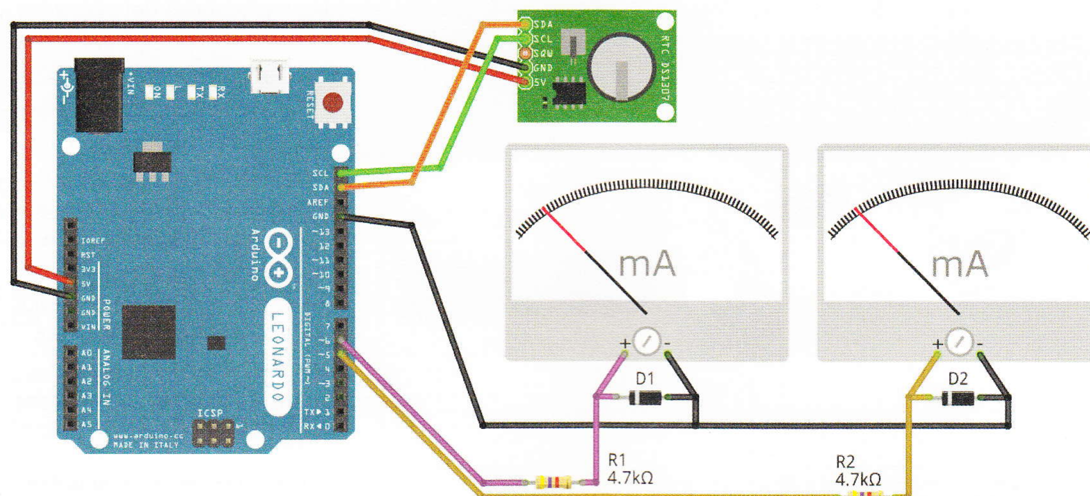
Les modules RTC comme celui-ci, un peu anciens, sont très courants et fournissent une base de temps pour de nombreux projets. L'élément actif est le circuit intégré DS1307, cadencé par un oscillateur à quartz et dont le fonctionnement est assuré soit par l'alimentation 5V, soit par la pile bouton CR2032.

- Deux ampèremètres analogiques, l'un pour l'heure, l'autre pour les minutes. Idéalement, il s'agira de versions 0-1 mA. J'en ai trouvé pour moins de 5 euros pièces (port gratuit) sur eBay auprès d'un vendeur appelé « *xyfs-us* » sous la désignation « *DC 0-1mA Analog Amp Meter Ammeter Current Panel New* ». La transaction s'est passée sans le moindre problème et la livraison depuis Shanghai (Chine) a été réceptionnée dans les 10 jours.

- Deux diodes 1N4148 : nous travaillons avec des bobines et avons donc à prendre en charge un problème inhérent à leur fonctionnement, qu'on retrouve dans l'utilisation des moteurs et des relais. Ces diodes sont ici des éléments de protection permettant d'éviter qu'une surtension n'endommage la carte Arduino lorsque nous coupons l'alimentation de la bobine. Étant donné le faible courant en présence et l'utilisation d'une résistance relativement conséquente, il est plus que probable que cette protection soit superflue. Il s'agit cependant, par principe, d'une bonne habitude à prendre lorsqu'on souhaite contrôler ainsi une bobine (inductance).

- Deux résistances de 4,7 Kohms : ces deux simples éléments sont là pour limiter le courant qui traverse les ampèremètres analogiques à 1 mA. Encore une fois, cette valeur est dépendante de la tension utilisée et de la valeur maximum des ampèremètres analogiques. Si vous mettez en œuvre une autre carte ou utilisez des ampèremètres de 10 mA, il faudra recalculer les valeurs.

LE MONTAGE



Avec une carte Arduino, nous n'avons pas de sortie analogique (véritablement analogique) et quand bien même nous disposerions de cela, il s'agirait d'un contrôle de tension, non de courant (avec un DAC intégré par exemple). Ce que nous pouvons faire en revanche consiste à utiliser une simple résistance pour limiter le courant à une valeur donnée, correspondant au maximum de ce que mesure l'ampèremètre analogique.

Si nous portons notre choix sur un ampèremètre analogique 0-1 mA (souvent simplement noté 1 mA), le calcul est simple : non seulement nous savons qu'une carte Arduino (Uno, Mega, Leonardo) peut fournir jusqu'à 40 mA (20 mA recommandé) par sortie (avec un maximum de 150 mA pour tout un port), mais nous connaissons également la tension en présence (5 volts).

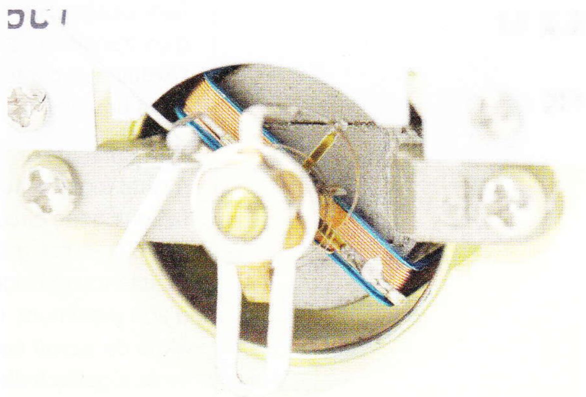
Pour calculer la valeur de notre résistance, il ne nous en faut pas plus, la loi d'Ohm est là pour cela :

```
U = R * I
5 volts = R * 0,001 A
5/0,001 = R = 5000 ohms = 5 Kohms
```

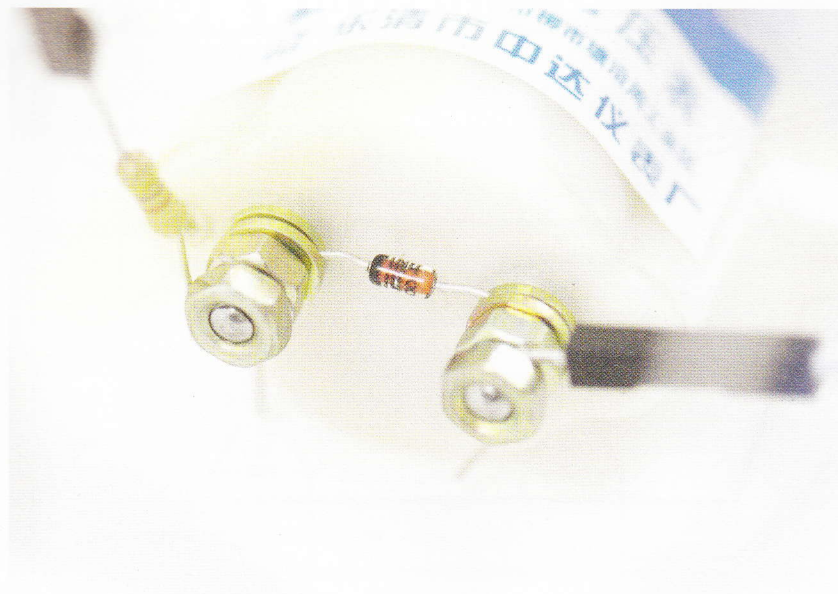
Cette valeur de 5 Kohms ne fait pas partie des séries de valeurs standardisées (E6, E12, E24, etc.). 4,7 Kohms en revanche est une valeur très commune, facile à trouver. En faisant rapidement le calcul inverse, nous obtenons :

```
U = R * I
5 = 4700 * I
I = 5/4700 = 0,00106 A = 1.06 mA
```

60 µA de plus que la valeur maximum de l'ampèremètre analogique n'est vraiment pas grand-chose, nous ne risquons pas d'endommager un élément aussi robuste qu'une simple bobine sur un ressort et sommes très loin des valeurs maximums d'une carte Arduino. 4,7 Kohms fera donc très bien l'affaire.



Gros plan sur la partie « active » de l'ampèremètre : une bobine montée sur un support mobile retenu par un ressort entourant un aimant permanent. Lorsqu'un courant passe dans la bobine, le champ magnétique produit va tenter d'aligner la bobine sur celui de l'aimant et l'aiguille change de position.



La diode dite de roue libre permet d'éviter une surtension lors de la coupure de l'alimentation de la bobine et de l'effondrement du champ magnétique. Ici, avec les courants en œuvre, le risque d'endommager le port de la carte Arduino est presque négligeable, mais mieux vaut prévenir que guérir, surtout pour le prix d'une simple diode 1N4148.

Notez qu'il existe des ampèremètres analogiques de toutes valeurs. Nos calculs le montrent clairement, 1 mA est la solution idéale, mais les versions 10 mA conviendraient également (avec une résistance de 470 ohms). Au-delà (100 mA et plus), une carte Arduino ne peut plus alimenter directement l'élément et il faudra faire usage d'un MOSFET ou d'un transistor, qui sera piloté par l'Arduino et contrôlera l'alimentation de la bobine.

À ce stade, en mettant la sortie à l'état logique haut (+5V) nous faisons donc circuler un courant limité à ~1 mA grâce à notre résistance et l'aiguille se déplace à son maximum. La question est alors de savoir comment placer cette aiguille à d'autres endroits que zéro et le maximum. La réponse s'appelle PWM bien sûr, et elle est déjà intégrée dans les cartes Arduino sous la forme de sorties (pseudo) analogiques.

Reportez-vous au premier numéro du magazine pour des explications détaillées sur la PWM. Rappelons toutefois rapidement le principe général : la PWM consiste à découper une tranche de temps en une série d'états haut ou bas. Si la totalité des tranches est à l'état haut, on dit qu'on a un rapport cyclique de 100%, la sortie est toujours à l'état haut. Si précisément la moitié des tranches est à l'état haut et l'autre à l'état bas, nous avons un rapport cyclique de 50%.

Dans ce genre de situation, certains composants se comportent comme s'ils étaient alimentés à 50% de leur capacité. C'est le cas des leds par exemple dont la luminosité varie en fonction du rapport cyclique. Chose impossible à obtenir en faisant varier le courant puisque les leds sont des composants qui demandent un courant fixé par le fabricant et qui fait apparaître une tension connue à leurs bornes.

La PWM peut être utilisée ainsi pour énormément de choses, dont le contrôle de la vitesse d'un moteur à courant continu. Qui dit « moteur », dit « bobine » et « aimant », choses qui ne sont pas très éloignées de nos galvanomètres (ou ampèremètres analogiques). Nous pouvons donc avec les sorties PWM d'une carte Arduino, régler un rapport cyclique et par la même occasion placer l'aiguille à 10%, 25%, 50%, etc., du maximum que nous avons d'ores et déjà fixé.

LE CROQUIS

Fichier Édition Croquis Outils Aide

```
#include <Wire.h>
#include <Time.h>
#include <DS1307RTC.h>

#define HEURES  6
#define MINUTES 5

void setup() {
  // les deux broches en sortie
  pinMode(HEURES, OUTPUT);
  pinMode(MINUTES, OUTPUT);
}

void loop() {
  // pour le stockage de l'heure
  tmElements_t tm;

  // puis-je lire l'heure ?
  if(RTC.read(tm)) {
    // oui
    // Si l'heure est entre 0 et 11
    if(tm.Hour < 12)
      // on utilise directement la valeur
      analogWrite(HEURES, 255/12*tm.Hour);
    else
      // sinon on est l'après-midi et on retranche 12
      analogWrite(HEURES, 255/12*(tm.Hour-12));
    // On utilise la valeur pour les minutes directement
    analogWrite(MINUTES, 255/60*(tm.Minute));
  } else {
    // On ne peut pas lire l'heure
    if (RTC.chipPresent()) {
      // heure n'est pas valide
      // il faut d'abord régler l'horloge avec le croquis
      // d'exemple TimeRTCSet
    } else {
      // problème avec DS1307
      // il faut vérifier les connexions
    }
  }
  // pause de 5s
  // il n'est pas utile de rafraîchir l'heure plus souvent
  delay(5000);
}
```

Arduino

À PROPOS DU CROQUIS

Le croquis est très simple et court. On prendra soin avant toutes choses d'installer les bibliothèques *DS1307RTC* et *Time* de Michael Margolis directement depuis le gestionnaire de bibliothèques de l'IDE Arduino. Si le module RTC est neuf ou que sa pile a été retirée, il conviendra également de définir l'heure en chargeant le croquis `TimeRTCSet` livré parmi les exemples de la bibliothèque *DS1307RTC*. Ceci aura pour effet de régler l'heure automatiquement sur celle du PC où a été compilé l'exemple.

Ce n'est qu'alors qu'il sera possible d'utiliser notre croquis qui se contente de lire l'heure fournie par le module RTC. Pour stocker cette information, nous utilisons un type de variable `tmElements_t`. C'est une structure définie dans `Time.h` contenant une série d'entiers sur 8 bits représentant respectivement les secondes, les minutes, les heures, le jour de la semaine (dimanche = 1), le jour dans le mois, le mois et le nombre d'années écoulées depuis 1970 (oui, on aura un problème dans 210 ans).

Pour lire les informations et les placer dans notre variable `tm`, nous utilisons la fonction `RTC.read()`. En fonction de la valeur retournée, nous pouvons déterminer si l'opération a réussi ou non (!= 0). En cas de succès, il nous suffit ensuite d'une petite règle de 3 pour obtenir une valeur entre 0 et 255 que nous pouvons directement utiliser avec `analogWrite()` pour régler le rapport cyclique.

Afin de rendre le cadran de l'heure plus lisible, nous nous limitons à l'heure civile sur 12 heures, mais la valeur placée dans `tm.Hour` (le membre `Hour` de notre variable `tm`) est sur 24 heures. On teste alors simplement si la valeur est supérieure à 11 (<12 donc) et le cas échéant on retranche simplement 12 à la valeur avant la règle de 3.

Si `RTC.read()` retourne zéro en revanche, c'est qu'il y a un souci avec la lecture du module RTC et nous devons éventuellement traiter le problème. On utilise alors `RTC.chipPresent()` qui retourne une valeur non nulle si le DS1307 est bien détecté. Si tel est le cas, on en conclut que l'heure lue est invalide, ce qui est typique d'un DS1307 neuf ou dont la pile a été retirée. Si `RTC.chipPresent()` retourne zéro, la bibliothèque ne peut pas communiquer avec le circuit intégré, il est soit défectueux, soit mal connecté.

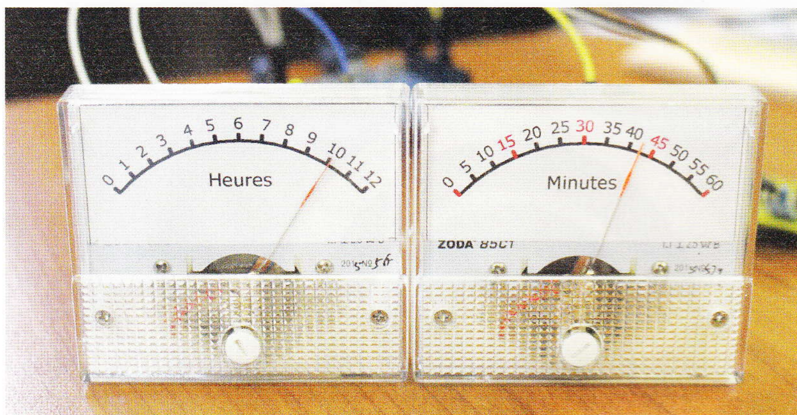
Notez que ces quelques lignes de code ne font strictement rien dans notre croquis et sont laissées à votre discrétion. Vous pouvez, comme ici, ne pas agir ou ajouter quelques lignes pour faire clignoter une ou plusieurs leds, ou encore agiter les aiguilles pour informer du problème. L'option du moniteur série est également envisageable, mais le fait de faire fonctionner ce montage à côté d'un PC qui affiche l'heure n'a pas vraiment de sens...

CONCLUSION ET SURTOUT... PERSPECTIVES !

Une modification des cadrans des ampèremètres analogiques est également un choix qui vous revient. L'opération n'est pas très difficile puisque, avec les modèles que j'ai utilisés, ceci revient simplement à retirer quatre vis (avec un cadran thermocollé, ce serait plus problématique). Il faut en revanche faire très attention, car le « mécanisme » interne est fragile. Mais en faisant preuve de prudence, on arrive à extraire le fond du cadran, prendre quelques mesures, passer un peu de temps avec Inkscape et enfin imprimer et coller sa composition avant de tout remonter proprement (comprendre « sans avoir de la colle plein les doigts »).

Bien entendu, avec un peu d'entraînement, vous pouvez aussi vous habituer à lire l'heure en laissant les cadrans intacts et en laissant ainsi planer le mystère quant à leur lecture et leur utilité.

Mais l'aspect le plus intéressant de ce montage est sans le moindre doute sa simplicité doublée de polyvalence. En effet, s'il est possible d'afficher l'heure de cette manière, il en va de même pour énormément de choses : une température, une intensité lumineuse, un volume sonore... Et en réfléchissant à l'ajout d'une liaison série avec un PC envoyant les valeurs à afficher, toute une série de réalisations s'ouvre à nous : occupation du disque dur, débit internet, charge processeur, utilisation de la mémoire, température dans le boîtier, nombre de mails non lus... Ceci sera sans doute plus facile à réaliser sous GNU/Linux, mais il doit exister des outils Windows pour communiquer les informations sur l'état du

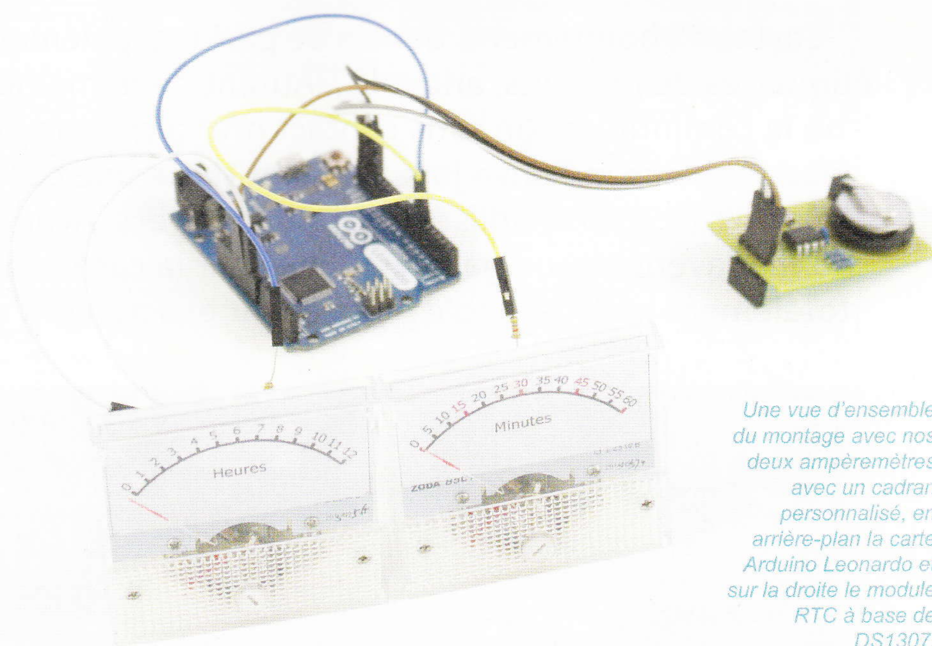


Il faut un peu de patience et de soin, mais arriver à dessiner un nouveau cadran en utilisant les bons angles et les bonnes proportions est tout à fait possible. Cela rend la lecture de l'heure bien plus facile et sera réutilisable pour d'autres applications du montage.

système via le port série et donc un moyen de les interpréter avec une carte Arduino (je n'ai pas poussé outre mesure dans ce sens, les termes « Windows » et « série » ou « serial » ayant une fâcheuse tendance à retourner n'importe quoi lors d'une recherche sur le Web).

Il est également possible d'envisager la traduction de ce montage sur Raspberry Pi. En utilisant un script Python avec le module `RPIO.PWM`, le principe de fonctionnement sera exactement le même, le rapport cyclique correspond à la position de l'aiguille sur le cadran. Attention cependant, la Raspberry Pi utilise des niveaux de tension de 0-3,3 volts. Il faudra donc adapter la valeur des résistances limitant le courant et ne pas oublier que le maximum est de 16mA pour une broche de la Pi et 51 mA sur l'ensemble des sorties.

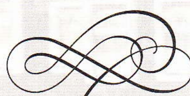
Une autre voie pratique et pédagogique intéressante consiste à chercher à porter notre croquis vers le minuscule Atmel ATtiny85 (ou 45). Ce microcontrôleur peut être programmé via l'environnement Arduino comme nous l'avons vu dans le numéro 8 du magazine. Il dispose de deux sorties PWM ainsi qu'une interface i2c. Il est donc parfaitement utilisable pour remplacer une carte UNO ou Leonardo dans ce cas de figure. La difficulté se résumera à l'adaptation du croquis, car les bibliothèques *Wire*, *Time* et *DS1307RTC* doivent laisser place à *TinyWireM* et *TinyRTCLib* (deux bibliothèques modifiées et distribuées par AdaFruit, basées sur des codes écrits respectivement par BroHogan et JeeLabs). Ces deux éléments doivent être installés dans l'IDE Arduino, tout comme le support pour les plateformes ATtiny (de David A. Mellis par exemple), et ne sont pas directement compatibles avec *Wire*, *Time* et *DS1307RTC*. Une bonne partie du croquis devra donc être réécrit, mais ceci peut être un excellent exercice pour explorer le domaine des tout petits microcontrôleurs tout en réduisant drastiquement le coût et l'encombrement de votre horloge. **DB**



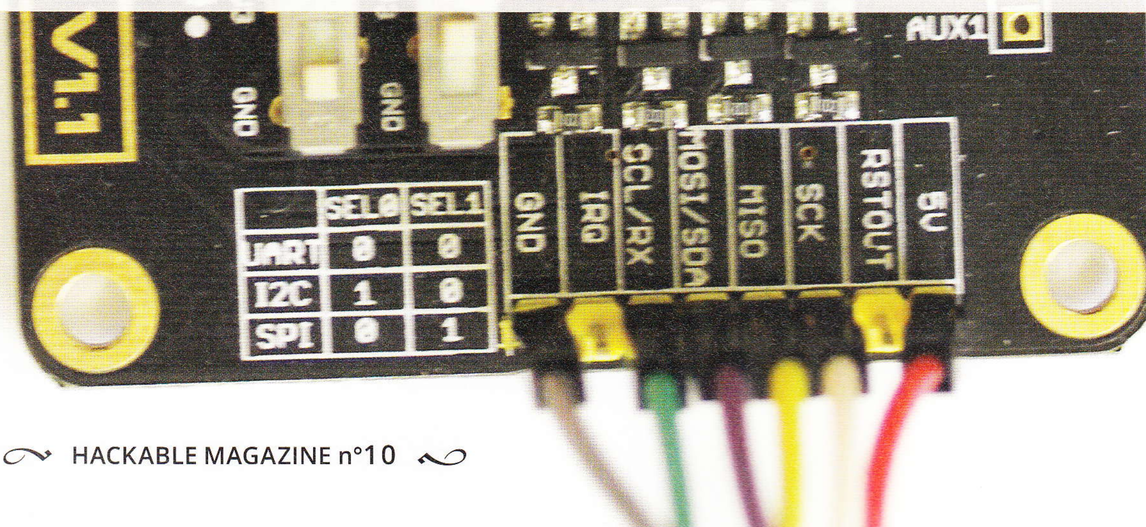
Une vue d'ensemble du montage avec nos deux ampèremètres avec un cadran personnalisé, en arrière-plan la carte Arduino Leonardo et sur la droite le module RTC à base de DS1307.

DÉCOUVREZ LA NFC ET LES TAGS RFID : LE GROS MINIMUM À SAVOIR

Denis Bodor



Cartes d'abonnement, tickets de parking, paiement sans contact, étiquettes de produits, antivols... Autant de domaines et d'applications où la communication sans contact prend chaque jour davantage de place et où se mélange joyeusement toute une collection de termes, parfois (souvent) utilisés à tort et à travers. Avant de nous lancer dans l'aventure, un passage obligé par la case « introduction » est totalement indispensable pour savoir exactement de quoi on parle !



Le mot « NFC » s'est depuis quelques années bien installé dans le langage courant, ainsi que « RFID ». Il est important en revanche de ne surtout pas tout mélanger, car il s'agit bien de deux choses distinctes même si technologiquement voisines et s'empruntant mutuellement certaines caractéristiques. Le but de cette introduction est de vous permettre de clairement distinguer quelles technologies sont désignées lorsqu'on évoque des termes comme NFC, RFID, ISO14443, NDEF, Mifare, NTAG... Le tout sans avoir à lire des centaines de pages de documentation ou spécifications de normes complexes.

1. LA BASE : LE RFID

Passons tout d'abord les fantasmes de type « les boîtes de raviolis RFID vont nous pister où qu'on aille » et penchons-nous surtout sur ce qu'est effectivement cette technologie. À la charge de chacun, ensuite, fort de connaissances techniques non imaginaires, d'avoir une réflexion personnelle et d'en tirer des conclusions concernant l'impact sur sa propre vie.

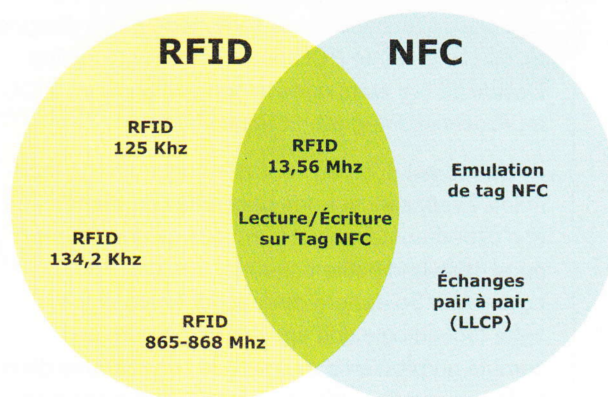
RFID pour *Radio Frequency IDentification* est un ensemble de normes et standards définissant une technologie permettant de mémoriser des données sur un support et de les faire transiter à distance sans connexion matérielle. Le plus souvent, ceci prend la forme de **marqueurs** désignés par les termes **tags RFID**, **radio-étiquettes** ou **transpondeur RFID**, pouvant se matérialiser de différentes façons : badges, autocollants, porte-clés, capsules sous-cutanées, cartes, disques plastiques à coller ou visser, ou encore embarqué directement dans un objet comme un outil, un jouet, un appareil domestique ou une peluche.

L'objectif initial de ces objets est avant tout de permettre une identification, mais comme nous allons le voir, dans les faits, les usages vont bien au-delà. Tout ce qui répond à cette définition peut être qualifié de technologie RFID, sous certaines conditions. L'une d'entre elles concerne la fréquence en œuvre qui peut être (liste non exhaustive) :

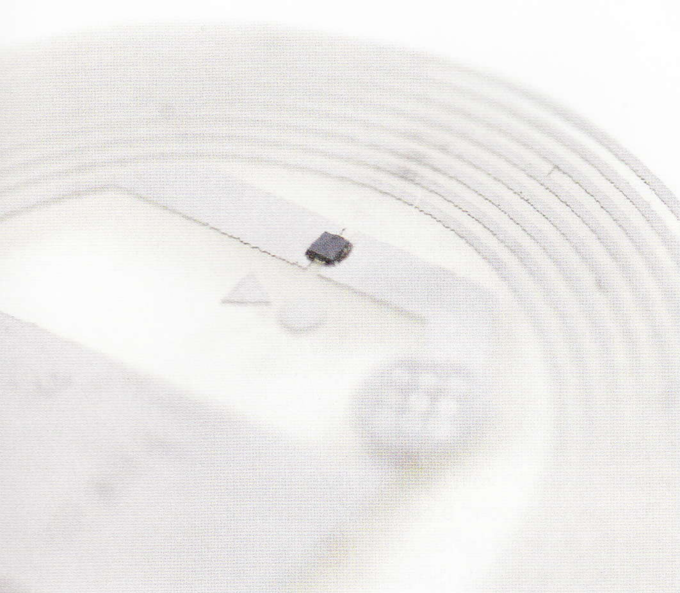
- Basse fréquence ou LF (*Low Frequency*) : à 125 ou 134 Khz ;
- Haute fréquence ou HF (*High Frequency*) : généralement 13,56 Mhz ;
- Ultra-haute fréquence ou UHF (*Ultra High Frequency*) : 865 MHz à 868 MHz (CE), et tantôt 2,4 Ghz ;
- Supra-haute fréquence ou SHF (*Super High Frequency*) : 5,8 GHz (généralement réservée pour les applications nécessitant une portée importante).

Ces fréquences sont utilisées pour deux choses : transmettre des informations et alimenter ce que nous appellerons désormais le tag RFID ou simplement le tag. J'écarterai de mes explications les tags actifs et semi-actifs alimentés par ailleurs et de plus en plus rares (sauf lorsqu'on parle de pair-à-pair). Ce qui nous intéresse ici, ce sont les tags dits passifs.

Ceux-ci sont constitués de deux principaux éléments : une antenne et une puce électronique (ou *die* en anglais). La puce est alimentée par induction électromagnétique selon un principe connu depuis plusieurs générations. Prenez une bobine, faites-y



RFID et NFC sont deux choses distinctes, reposant sur des normes et des standards bien définis. La première chose à savoir sur ces technologies consiste à de ne pas les confondre et ne jamais utiliser un terme à la place de l'autre.



Ce petit carré noir de moins d'un millimètre de côté est le circuit intégré formant la partie intelligente d'un tag (ici un NXP NTAG213). Celui-ci est alimenté par l'antenne en spirale qui l'entoure et lui permet également de communiquer.

passer un courant alternatif et placez une autre bobine à proximité. Le champ magnétique apparaissant et disparaissant dans la première bobine va induire un courant alternatif dans la seconde. C'est exactement le même principe de fonctionnement qu'un transformateur, à la différence que les deux bobines ne sont pas fixes et que le champ magnétique s'étend dans l'air et non dans une carcasse (ou tôles feuilletées). Le procédé a fait l'objet de nombreuses expérimentations très popularisées dès la fin du 19ème siècle par un certain... je vous le donne en mille... Nikola Tesla ! Les proportions en œuvre étaient cependant toutes autres que celles des tags RFID (cherchez « tour Wardenclyffe » pour vous faire une idée). C'est également ce même principe qui est utilisé avec les smartphones modernes proposant un système de recharge sans fil.

L'énergie ainsi transmise, qui est modulée par l'émetteur, permet d'alimenter une puce dédiée à cet usage qui en retour va pouvoir répondre en réfléchissant une partie du rayonnement électro-magnétique (phénomène de rétrodiffusion) ou en l'absorbant. Nous avons donc une communication bidirectionnelle, *half duplex* (HDX) ou réception puis envoi pour le tag, et *full duplex* ou envoi et réception simultanés pour l'émetteur, qu'on désigne plutôt par le terme PCD pour *Proximity Coupling Device* (le tag étant dans la nomenclature un PICC ou *Proximity Integrated Circuit Card*).

Un tag est donc un véritable concentré de technologie reposant sur des décennies de recherches et d'expérimentations et bien plus qu'une simple puce dans une petite étiquette, une carte ou un porte-clés. En dehors de l'aspect énergétique, cette technologie doit également prendre en compte un problème majeur que l'on retrouve dans la documentation désigné par le terme « collision ». En effet, lorsque plusieurs tags (PICC) se trouvent dans le champ du lecteur (PCD) les communications se mélangent et des techniques particulières doivent être utilisées pour contourner ces effets : les algorithmes d'anti-collision.

Les fréquences, les techniques de modulation de signaux, les algorithmes de communication et d'anti-collision et bien d'autres choses encore, forment ensemble une série de normes et de standards. C'est cet ensemble qui est désigné sous le terme RFID.

2. LA NORME : LE NFC

Les mots « NFC » et « RFID » ne sont pas interchangeables. Utiliser maladroitement l'un à la place de l'autre est tout à fait comparable au fait de confondre CD et DVD.

La technologie NFC pour *Near Field Communication* ressemble au RFID, mais n'en est pas, car encadrée par d'autres standards et normes (regroupés sous les désignations ISO/IEC 14443-1 à ISO/IEC 14443-4). Ces normes décrivent quelque chose s'approchant du RFID, mais imposent un certain nombre de restrictions tout en couvrant des points ignorés par les normes RFID. Ceci concerne les caractéristiques physiques (14443-1), l'interface d'alimentation et de gestion radiofréquence (14443-2), l'initialisation et l'anti-collision (14443-3) et enfin le protocole de transmission (14443-4).

Ne vous inquiétez pas, si vous ne comptez pas fabriquer de lecteurs ou de tags, nous n'avez pas

besoin de comprendre le contenu de ces normes et ces caractéristiques. En revanche, savoir qui fait quoi sous quelle désignation est absolument indispensable pour ne pas faire d'erreur et se retrouver dans une impasse lors de vos expérimentations.

Ceci commence par la fréquence utilisée qui, en NFC, se bornera à 13,56 Mhz. Les tags LF RFID 125 Khz ou 134,2 Khz (utilisés pour le marquage des animaux) ne sont donc **PAS** des tags NFC.

Pour communiquer sur cette fréquence, le lecteur et le tag utilisent une modulation. Deux types de modulation sont couverts par la norme sous les noms ISO 14443-A et 14443-B ou plus simplement des lecteurs type A ou B et des tags type A ou B. Bien entendu, un lecteur type A ne pourra pas communiquer avec un tag type B et inversement. Il est donc important en choisissant votre matériel de prendre ceci en considération et d'opter pour un périphérique capable de gérer les deux types (ainsi que d'autres dont nous parlerons plus loin). Notez qu'il existe également un type F, presque exclusivement utilisé au Japon (cartes Suica pour le réseau ferroviaire).

Le type de modulation A ou B est indépendant du protocole (décrit par 14443-4) qui est grossièrement la façon de communiquer. Un peu comme pour des échanges entre personnes : le type est la langue utilisée, le protocole est « bonjour, question, formule de politesse ».

Voilà pour ce qui est de la communication et des protocoles, mais ce n'est pas tout. Parmi les tags, il faut également distinguer différents types en fonction des fonctionnalités qu'ils proposent :

- Le type 1 dispose d'un UID (*Unique IDentifier*), un numéro unique par tag, assimilable à un numéro de série, mais valable parmi la totalité des tags produits tous fabricants confondus. Ce type est également verrouillable en lecture seule. Le tag Topaz d'Innovision (maintenant Broadcom) est un exemple de tags de type 1.



Une petite partie d'une collection assez classique de tags de toutes sortes, Mifare Classic, Mifare Ultralight, NTAG203, NTAG213, Mifare DESFire... et de toutes formes, cartes, stickers, porte-clés...

- Le type 2 possède l'UID, est verrouillable et ajoute l'anti-collision permettant de solutionner le problème de la lecture simultanée de plusieurs tags. Les Mifare Ultralight et Ultralight C ainsi que NXP NTAG213 par exemple sont des tags de type 2.
- Le type 3 n'a pas d'UID, mais est verrouillable en lecture seule et dispose des mécanismes anti-collision. Le Sony FeliCa est de type 3.
- Le type 4 est l'aîné de la famille avec l'UID, le verrouillage, l'anti-collision et la possibilité d'avoir un contenu actif, le tag lui-même peut modifier son contenu (et non simplement le PCD). Un tag NXP DESFire est une implémentation d'un tag de type 4.

Enfin, un dernier point important qui fait que la technologie en œuvre peut être qualifiée de NFC concerne le mode de communication. Le standard décrit, pour un lecteur (PCD), deux modes possibles. Le premier est celui que j'ai détaillé dans la partie RFID où la communication est initiée par un périphérique et l'autre répond en modulant le champ magnétique fourni par l'initiateur. Le second mode est actif et le champ magnétique est produit alternativement de part et d'autre de la communication, chaque périphérique désactivant son émission pour recevoir des données. Ce second mode permet par exemple à deux smartphones disposant de fonctionnalités NFC de s'échanger des informations, l'un jouant le rôle de lecteur (initiateur) et l'autre de tag (cible). Ces échanges peuvent avoir lieu de pair à pair (LLCP pour *Logical Link Control Protocol*) ou avec un des périphériques émulant un tag.

Les tags peuvent prendre la forme de cartes au format standard. Il est même possible de combiner plusieurs technologies avec ici à droite une carte intégrant un tag Mifare DESFire, une puce JCOP (JavaCard) et, au dos, une piste magnétique HiCo.



Il faut bien comprendre que les produits, objets et tags disponibles, doivent être vus comme des « implémentations ISO/IEC 14443 » ou répondant à cette norme, de manière totale ou partielle. Quelques exemples d'implémentation sont le passeport biométrique, les cartes de transport type Calypso (qui sont de type B', un type B mais avec un protocole propriétaire), la carte d'identité allemande, les cartes de paiement sans contact EMV (*Europay MasterCard Visa*) et bien entendu, les périphériques et tags officiellement compatibles NFC, selon le *NFC Forum*.

Ces considérations de compatibilité et désignations associées sont certes pénibles et peu enjouantes, mais c'est le bagage minimum pour éviter les écueils. À titre d'exemple, pour en revenir à la thématique propre du magazine, certains modules ou shields pour Arduino sont capables de lire toutes sortes de tags, mais ne sont pas compatibles NFC.

C'est le cas des modules construits autour de la puce RC522 (MFRC522 de NXP plus

précisément), car celle-ci est en réalité un composant pour **lecteur** de tags NXP Mifare (Classic, Mini, Ultralight, DESFire, etc.) et ne peut agir qu'en tant qu'initiateur et non de cible. Ce n'est donc pas une solution NFC, mais juste un périphérique pour lire et écrire les tags RFID NXP Mifare. Il ne faut donc pas s'étonner du fait que ce composant ne soit pas supporté par des bibliothèques permettant le support NFC (comme les NFC-tools pour la Pi) alors que le PN532, également à la base de certains modules et shields et du même fabricant, le sera parfaitement.

3. NFC C'EST AUSSI UN FORMAT DE DONNÉES

Pour résumer, nous avons des objets qui utilisent des fréquences modulées pour communiquer selon certains protocoles, le tout classé proprement par types et libellé de tout un tas de désignations cryptiques (et encore, je vous en ai épargné la plupart). C'est un peu comme quand on range son bureau, si déjà on décide de mettre de l'ordre et d'organiser, autant le faire totalement. Le NFC Forum ne s'est donc pas limité à l'aspect matériel, mais a également défini un format pour les données stockées ou échangées.

La façon dont les données sont stockées physiquement dans un tag NFC est dépendant du tag lui-même, de la technologique utilisée, du modèle et du constructeur. Certains tags utilisent des blocs de 4 octets appelés pages, d'autres des secteurs de 4 blocs de 16 octets et d'autres encore une architecture qui rappelle les fichiers d'un disque dur. Pour que tout le monde puisse échanger des données convenablement, se place au-dessus du stockage physique un format unique : NDEF pour *NFC Data Exchange Format*, ou en français, un format d'échange de données NFC.

Ce format décrit des **messages** NDEF constitués d'un ou plusieurs **enregistrements** NDEF. Ces enregistrements ne contiennent pas uniquement les données utiles (*payload* dans le jargon), mais bien d'autres choses comme le type de type d'enregistrement (non ce n'est pas une faute de frappe, c'est le TNF pour *Type Name Format*), si c'est un morceau d'un enregistrement (CF), s'il s'agit du début (MB) ou la fin (ME) d'un message, la taille des données utiles, le type d'enregistrement (en fonction du TNF), éventuellement un numéro... et finalement les données utiles.

C'est la même logique que celle d'un format graphique dans lequel on trouve non seulement la liste des couleurs de chaque pixel, mais également toutes les informations permettant de traiter l'image en tant que telle (taille, densité, palettes, métadonnées, etc.). Ici, c'est simplement un peu plus riche et générique, car un tag NFC peut contenir énormément de choses différentes.

À titre d'aperçu, prenons un cas particulier avec un enregistrement ayant le champ TNF à `0x01`. Ceci indique qu'il s'agit d'un enregistrement de « type bien connu » (ça sonne mieux en anglais : « *Well-Known Type* » ou WKT). De ce fait, le type précisé dans l'enregistrement aura un sens précis et standardisé où chaque valeur possible fait partie d'une nomenclature appelée RTD (*Record Type Definition*). Si nous prenons le type `0x55` (qui correspond au caractère `U`), l'enregistrement est utilisé pour stocker un URI ou *Uniform Resource Identifier*, autrement dit une chaîne définissant quelque chose. Et ce quelque chose peut, par exemple, être une adresse mail (`0x06` pour `mailto:`)...

Si là vous n'avez pas encore décroché, félicitations ! Mais sachez que cet exemple ne concerne qu'un TNF (`0x01`), un type d'enregistrement (`0x55`) et un code URI (`0x06`). Il y a une quantité de codes URI, une demi-douzaine de « types bien connus » (WKT) et 7 TNF (types de type). S'ajoute encore à cela le TNF `0x04` précisant qu'une « nomenclature » externe (*NFC Forum External Type*) peut être utilisée et donc décrite en dehors des spécifications du NFC Forum.

Étant donné la complexité ou plus exactement la densité du sujet et la masse d'informations à assimiler, il n'est pas question pour un usage comme le nôtre de devoir lire et retenir des pages et des pages de spécifications. Exactement de la même manière qu'il ne vous est pas nécessaire de comprendre le format JPEG ou PNG pour manipuler des images

sur un PC ou un Mac, nous n'aurons pas besoin de maîtriser le format NDEF pour faire usage de la technologie NFC.

Comme pour les images, seuls quelques points importants sont à retenir et en particulier le fait qu'un tag NFC contient toujours des données NDEF qui sont davantage que de simples données utiles. Si un tag ne contient autre chose que du NDEF, ce n'est pas un tag NFC. Encore une fois, voyez cela comme un support optique : si un CD ne contient pas des données formatées CDDA (alias Red Book), ce n'est pas un CD audio, même s'il s'agit bien d'un CD et que de la musique y est enregistrée.

4. LES DIFFÉRENTS TAGS

À ce stade, vous pouvez vous demander ce qu'est finalement effectivement un tag NFC et la question est parfaitement légitime. La réponse est simple : c'est un tag répondant aux spécifications NFC et contenant des données formatées NDEF, rien de plus.

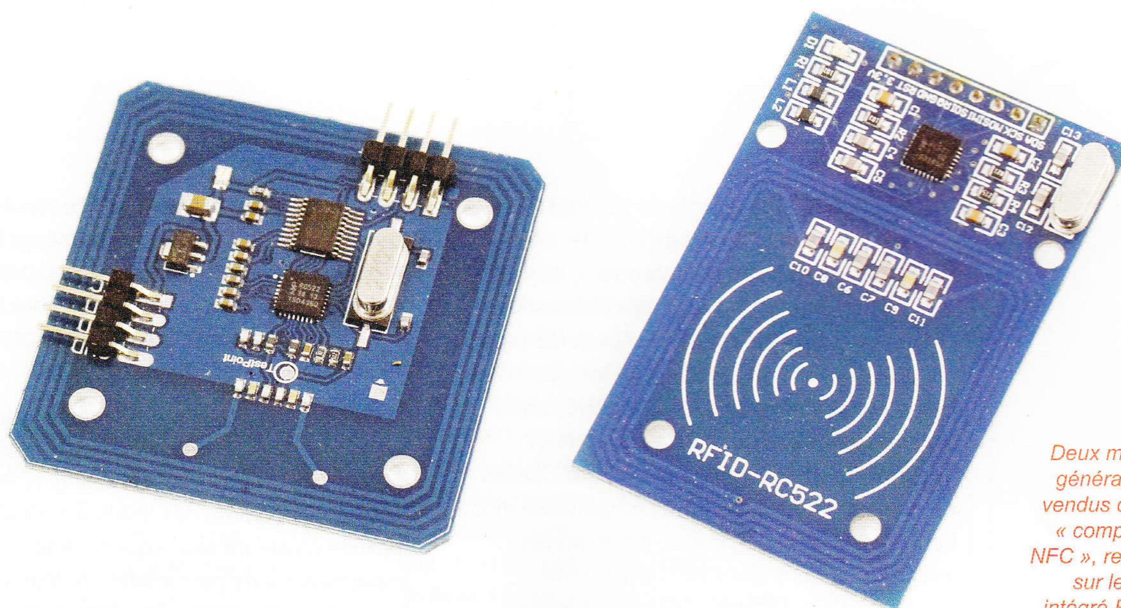
Matériellement, comme le disent clairement les documentations de NXP pour ces composants, c'est juste une puce compatible avec les normes définissant un tag type 1, 2, 3 ou 4. Et ceci ne signifie absolument pas que vous ne pouvez qu'y inscrire des données NDEF. Ainsi, ceux qu'on trouve généralement dans le commerce sont des tags ayant une désignation définie par leurs fabricants. Il n'y a pas de tags NFC « tout court ». Un NTAG213 et un Mifare Ultralight, de NXP sont des produits différents, mais tous les deux peuvent être des tags NFC type 2 s'ils contiennent les bonnes données.

Le NFC Forum ayant été formé par les principaux acteurs sur le marché que sont NXP, Sony, et Innovision/Broadcom, il n'est guère étonnant que les tags commercialisés et se trouvant un peu partout proviennent principalement de ces constructeurs avec, en tête de liste NXP. Sans oublier, bien entendu les fabricants de clones et de produits compatibles comme Infineon Technologies (ex-Siemens) ou Fudan Microelectronics (il existe même des tags produits sans licence, disposant de fonctionnalités hors normes, comme des UID réinscriptibles).

Pour expérimenter ces technologies, vous aurez besoin d'un lecteur (ou *reader* qui est aussi un périphérique d'écriture, mais c'est là la désignation courante, un peu comme les *lecteurs* de disquettes d'antan), mais aussi et surtout de tags, de formes et de modèles les plus variés possibles de préférence.

Ainsi, parmi les produits les plus courants nous avons :

- Les *Mifare Classic* de NXP qui ne sont pas à proprement parler des tags compatibles NFC (voir ci-après) mais ce sont, sans le moindre doute ceux qu'on trouve littéralement partout. Ceux-ci se déclinent en deux tailles, 1k (752 octets utiles) ou 4k (3440 octets utiles), parfois respectivement désignés par « S50 » et « S70 ».
- Les *Mifare Ultralight* également de NXP qui cette fois respectent intégralement les standards NFC et peuvent être utilisés dans des tags NFC de type 2. Initialement créés pour les applications à faible coût, on les retrouve généralement dans des solutions à durée de vie limitée, embarquées dans des supports peu robustes comme les tickets de parking.
- Les NTAG210/213/215/216 sont très proches des *Mifare Ultralight*, mais ont été développés spécifiquement par NXP pour un usage en tant que tags NFC type 2 de 48, 144, 504 et 888 octets utiles. On peut voir ce modèle comme le successeur de l'Ultralight pour les usages NFC.
- Les *Mifare DESFire* sont des tags plus évolués intégrant un microprocesseur et toute une logique interne ainsi qu'un système d'exploitation et différents mécanismes de protection avancés. Une évolution du produit est disponible sous



Deux modules généralement vendus comme « compatibles NFC », reposant sur le circuit intégré RC522. Au sens strict du terme, il ne s'agit pas de NFC puisque ce composant ne peut émuler un tag, mais plutôt d'un simple lecteur de tags NXP Mifare.

le nom *Mifare DESFire EV1* ajoutant encore des fonctionnalités et corrigeant certains problèmes de sécurité. Les DESFire sont totalement compatibles avec les normes du NFC Forum pour une utilisation en tag NFC de type 4.

Je pense qu'on peut parler ici d'une réelle domination de NXP, du moins pour l'Occident. En effet, Sony et sa technologie FeliCa, qui est compatible NFC (parce que la compatibilité FeliCa est incluse dans le standard), est massivement utilisé dans les pays asiatiques (Japon, Chine, Thaïlande, etc.) ainsi que de façon éparse pour certaines applications dans le reste du monde.

On trouve ensuite tout un tas d'autres tags pouvant ou non être compatibles NFC Forum, tantôt reposant sur des technologies connues sous un autre nom. C'est le cas par exemple des Samsung TecTiles, reposant sur NXP Mifare Classic. Mais on trouve également du Texas Instrument Tag-it HF Plus, du NTAG203, du MIFARE Plus, etc.

Mes recommandations concernant la composition d'une petite collection de tags tiennent en peu de choses : le bien nommé Mifare Classic est le tag le plus facile à trouver et le moins cher, il est donc presque indispensable (malgré ses écarts à la norme). Mais par souci de respect des normes, quelques Mifare Ultralight et NTAG213 (ou 203, 210, 215, 216) forment un complément pertinent. Et enfin par curiosité, on pourra tenter d'obtenir un ou des Mifare DESFire, FeliCa et autres modèles plus exotiques.

5. MIFARE CLASSIC ET LES PETITS ÉCARTS À LA NORME

Les tags Mifare Classic ne datent pas d'hier et étaient déjà utilisés avant même que l'on parle de NFC avec autant d'ardeur. La technologie utilisée est historiquement celle de la société Mikron (le nom MIFARE signifie d'ailleurs « *MIkron FARE Collection System* ») acquise par Philips, maintenant NXP, il y a presque 20 ans.

On retrouve les tags Mifare Classic sous une quantité incroyable de formes et de formats. C'est une technologie vastement installée, mais il ne s'agit pas véritablement de NFC. En effet, les Mifare Classic sont bel et bien des tags ISO/IEC 14443-A, mais ils ne respectent pas l'ensemble des spécifications du NFC Forum et en particulier ISO/IEC 14443-4 au bénéfice d'un protocole propriétaire propre aux puces du constructeur (et que partiellement ISO/IEC 14443-3).

La conséquence directe de cet état de fait est un problème de compatibilité. Le protocole utilisé étant non standard, seuls les composants NXP (ou compatibles sous licence) peuvent communiquer avec les tags Mifare Classic. Dans les faits si nous prenons l'exemple des périphériques Android, seuls les smartphones équipés de puces NXP peuvent lire et écrire des tags Mifare Classic en plus de ceux pleinement compatibles NFC. Les Google Nexus 4, 5 et 6, par exemple, intègrent un contrôleur NFC Broadcom (BCM20793 ou BCM20795) qui est compatible NFC Forum... et uniquement compatible NFC Forum. Ces smartphones ne permettent pas la manipulation de tags Mifare Classic, mais uniquement de tags qui sont également 100% compatibles NFC Forum (comme les NTAG, Mifare DESFire ou Mifare Ultralight).

Notez cependant que sur un smartphone équipé d'une puce NXP, il est tout à fait possible d'utiliser un tag Mifare Classic comme un tag NFC et donc de le formater et d'y placer des données NDEF. Tout fonctionnera exactement de la même façon et sera totalement transparent pour vous. Le tag en question, toutefois, ne sera lisible et donc utilisable que sur les smartphones également équipés d'une puce NXP.

Enfin, il est important de préciser que les Mifare Classic malgré les mécanismes de sécurité en place ne doivent plus être considérés comme des solutions sûres. Il existe plusieurs techniques, attaques et implémentations permettant de déjouer ces sécurités et d'accéder frauduleusement au contenu du tag.

Le « lecteur » ACR122U d'ACS est le périphérique USB le plus populaire et le plus facile à trouver, mais c'est aussi celui qui posera tantôt problème et obligera à une déconnexion/reconnexion en raison d'un firmware peu stable.



6. LE MATÉRIEL

Nous avons déjà parlé de la constitution minimale d'une collection de tags. Si le domaine vous apparaît divertissant après quelques essais, vous verrez sans le moindre doute votre collection rapidement s'étoffer de toutes sortes de choses (attention, il y a un petit effet Pokemon dans RFID/NFC). Ceci aussi bien au travers d'achats en ligne via des boutiques et sites d'enchères que par d'autres moyens. Il y a fort à parier que toutes les cartes dans votre portefeuille vont passer au test systématique, mais ce n'est pas tout...

Vous vous souvenez lorsque vos parents vous disaient qu'il ne fallait pas ramasser ce qui traîne par terre dans la rue ? Hé bien c'est faux ! Ne leur en voulez pas, il ne pouvaient pas savoir. Tantôt les tickets jetables embarquent une antenne et une puce NFC (de l'Ultralight le plus souvent), mais il peut aussi s'agir d'emballages divers, de cartes soi-disant non rechargeables, de badges égarés, de titres de transport, etc.

Mais le plus important n'est pas tant la collection de tags que le matériel permettant de les manipuler. La trousse à outils idéale, selon moi, est la suivante (par ordre d'importance) :

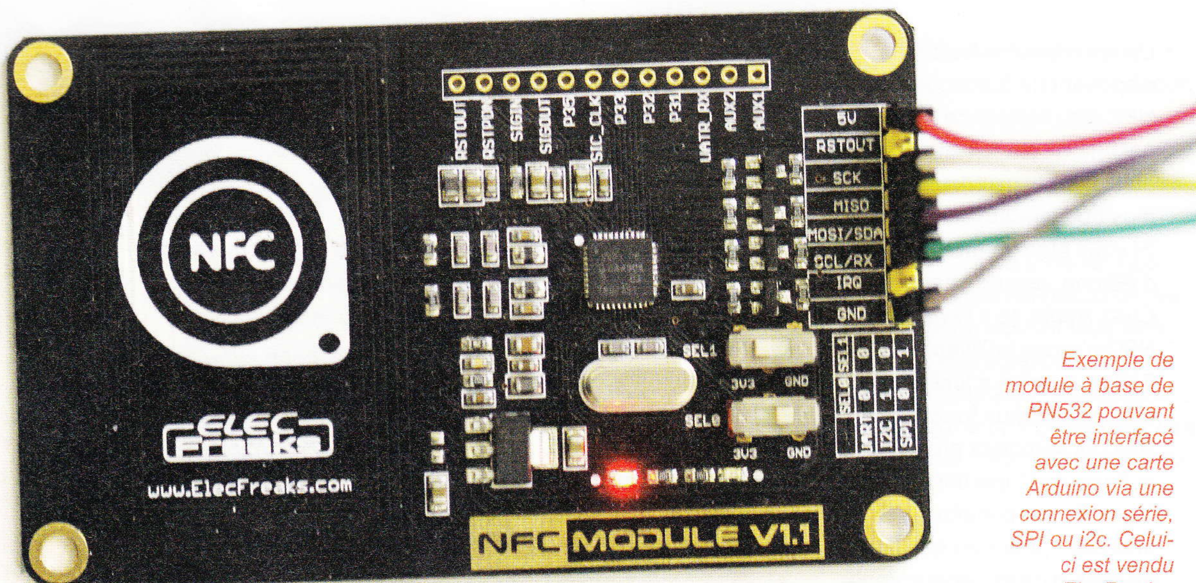
- Un smartphone Android disposant de fonctionnalités NFC, de préférence avec une puce NXP offrant un accès aux tags Mifare Classic en plus du standard NFC Forum. Si vous êtes un utilisateur d'iPhone, pas de chance. Chez Apple, la « révolution » NFC n'arrive qu'avec l'iPhone 6 et en version « limitée » uniquement aux fonctionnalités qui semblent profiter à la firme de Cupertino. L'offre logicielle sur plateforme Android est, pour peu que l'on dispose d'un appareil compatible, absolument fantastique. À titre personnel, bien qu'utilisant un non-smartphone Samsung E1200 (prix : 15€, autonomie : 1 mois), je conserve et chéris mon Samsung Galaxy Nexus et sa puce NXP PN544 pour cette raison précise (ainsi qu'une tablette Nexus 7 2012 intégrant une puce NXP PN65).
- Un « lecteur » NFC USB qui pourra être utilisé aussi bien sur PC/Mac qu'avec la Raspberry Pi. Deux lecteurs se partagent généralement la vedette avec d'un côté l'ACR122U d'ACS pour quelques 30€ et de l'autre le SCL3711 de SCM/Identive entre 40€ et 50€. Le premier est littéralement la solution la moins chère, mais présente tantôt un comportement instable bien connu. Le produit SCM/Identive est généralement un meilleur choix, plus stable et de bien moindre encombrement.
- Un module ou shield Arduino construit autour d'un PN532. Ce circuit intégré de chez NXP est compatible NFC et est donc capable non seulement de lire et d'écrire les Mifare Classic ainsi que les tags compatibles NFC (FeliCa inclus), mais également d'agir comme une cible NFC en pair-à-pair et en émulation. Le PN532 peut être interfacé en liaison série, SPI ou i2c. Il est donc important de choisir un module proposant de préférence les trois connectiques ainsi qu'une configuration des tensions (5V et 3,3V). De cette manière, un seul module vous suffira pour toutes vos expériences aussi bien sur Raspberry Pi que sur Arduino, Ti Launchpad, etc. Un tel module vous coûtera entre 15€ et 20€.
- Un module ou shield basé sur une puce RC522. Ce contrôleur RFID n'est **pas** compatible NFC au sens strict du terme puisqu'il ne sait agir qu'en tant qu'initiateur dans une transaction. Il ne pourra pas vous permettre un fonctionnement pair-à-pair avec un smartphone par exemple, mais sera bien utile avec une carte Arduino pour lire toutes sortes de tags (ISO/IEC 14443-A/Mifare). Comme le PN532, il peut être interfacé en série, SPI et i2c, et par-dessus tout, il présente un avantage crucial : un tel module coûte moins de 3€ !

En termes de budget, le smartphone ou la tablette idéalement compatible reste l'élément le plus coûteux s'il vous faut envisager une telle acquisition. Des périphériques d'occasion se trouvent entre 50€ et plus de 150€ selon l'état. Vous passer d'une puce NXP ne sera pas un problème, mais mettra les populaires tags Mifare Classic hors de votre portée. Le smartphone NFC Android n'est en rien indispensable, mais apporte un confort très important en faisant office de système portatif fiable, ne serait-ce que pour vérifier vos expérimentations sur Arduino ou Raspberry Pi.

Notez qu'il existe une solution permettant d'utiliser des lecteurs ACS avec une tablette Android disposant de fonctionnalités USB hôte. Ce support appelé « *External NFC Reader Service* », disponible via le



Le périphérique USB SCM SCL3711 est un produit compact et réputé en termes de stabilité de fonctionnement (par rapport à l'ACR122U). C'est le « lecteur » généralement recommandé si l'on souhaite explorer les technologies NFC et RFID 13,56 Mhz.



Exemple de module à base de PN532 pouvant être interfacé avec une carte Arduino via une connexion série, SPI ou i2c. Celui-ci est vendu par ElecFreaks, mais il en existe de nombreux modèles.

Play Store, est vendu 20€ ! Nous ne pouvons vous confirmer son fonctionnement, car nous ne l'avons pas testé pour une raison évidente : 30€ d'ACR122U + 20€ d'applicatif + le temps de développement (il ne s'agit pas d'une application, mais d'un service)... Il est plus simple et économique de se trouver un smartphone compatible d'occasion. Prenez simplement garde, certains modèles utilisent une antenne NFC placée dans la batterie (Galaxy Nexus) et les batteries « compatibles » et économiques ne l'intègrent pas.

Le lecteur USB lui est à envisager de la même manière, avec une souplesse un peu plus réduite puisqu'il ne disposera pas des applications clés en main et nécessitera l'ajout d'un PC/Mac (de préférence sous GNU/Linux) ou d'une Pi, ce qui reste handicapant pour une utilisation nomade. Un module PN532 pourra être connecté à un PC/Mac ou une Pi de façon à permettre une utilisation comparable à celle d'un lecteur USB, mais ceci demandera un peu de bricolage, soit par la connexion directe (série, SPI ou i2c), soit via un adaptateur USB/série.

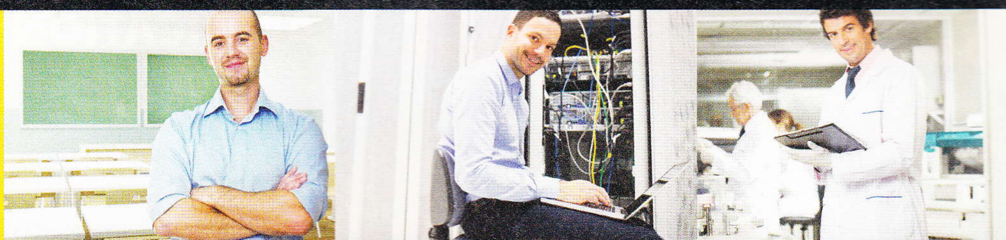
Comme dans toutes expérimentations, il est important d'avoir une référence sûre. Ainsi même si les modules à 3€ à base de RC522 sont fort séduisants, en cas de difficulté, vous ne saurez pas si le problème vient de vous, de votre code, du tag, du module, de la configuration... Vous serez dans le noir.

CONCLUSION

Longue, pénible et pas forcément très ludique... Voilà qui peut parfaitement qualifier une introduction aux technologies RFID et NFC comme celle que vous venez de subir. Mais cela reste un coût relativement modeste au regard de la richesse et de la rigidité des standards, mais aussi et surtout des possibilités qui vous sont maintenant accessibles. Nous ne sommes pas vraiment descendus dans le terrier du lapin blanc, car il y a énormément de choses à dire, en particulier sur le stockage des données sur différents types de tags. C'est là un autre avantage du NFC, il permet de se placer à un niveau d'abstraction où seules les données importent, le reste étant laissé à la charge des bibliothèques, des outils et des applications. Nous voici enfin prêts pour nous lancer dans l'aventure. Allons-y ! **DB**

PROFESSIONNELS DES TICE, COLLECTIVITÉS, ÉCOLES D'INGÉNIEURS, UNIVERSITÉS, R & D, ENSEIGNANTS, ...

HACKABLE MAGAZINE



PROFESSIONNELS !

DÉCOUVREZ NOS OFFRES D'ABONNEMENTS ...
...EN VOUS CONNECTANT À L'ESPACE DÉDIÉ AUX PROFESSIONNELS SUR :

www.ed-diamond.com

PDF COLLECTIFS PRO

OFFRE	ABONNEMENT	1 - 5 lecteurs Réf	1 - 5 lecteurs Tarif TTC	6 - 10 lecteurs Réf	6 - 10 lecteurs Tarif TTC	11 - 25 lecteurs Réf	11 - 25 lecteurs Tarif TTC
PROHK2	6n° HK	☐ PRO HK2/5	156,-	☐ PRO HK2/10	312,-	☐ PRO HK2/25	624,-

PROFESSIONNELS : N'HÉSITEZ PAS À NOUS CONTACTER POUR UN DEVIS PERSONNALISÉ PAR E-MAIL : abopro@ed-diamond.com OU PAR TÉLÉPHONE : 03 67 10 00 20

ACCÈS COLLECTIFS BASE DOCUMENTAIRE PRO OPEN SILICIUM

OFFRE	ABONNEMENT	1 - 5 connexion(s) Réf	1 - 5 connexion(s) Tarif TTC	6 - 10 connexions Réf	6 - 10 connexions Tarif TTC	11 - 25 connexions Réf	11 - 25 connexions Tarif TTC
PROOS+3	OS	☐ PRO OS+3/5	90,-	☐ PRO OS+3/10	180,-	☐ PRO OS+3/25	360,-
PROH+3	GLMF + HS + LP + HS + MISC + HS + OS	☐ PRO H+3/5	447,-	☐ PRO H+3/10	894,-	☐ PRO H+3/25	1788,-

Les abréviations des offres sont les suivantes : LM = GNU/Linux Magazine France HS = Hors-Série LP = Linux Pratique OS = Open Silicium

SÉLECTIONNEZ VOTRE OFFRE DANS LA GRILLE CI-DESSUS ET RENVOYEZ CE DOCUMENT COMPLET À L'ADRESSE CI-DESSOUS !

Voici mes coordonnées postales :	
Société :	
Nom :	
Prénom :	
Adresse :	
Code Postal :	
Ville :	
Pays :	
Téléphone :	
E-mail :	

☐ Je souhaite recevoir les offres promotionnelles et newsletters des Éditions Diamond.

☐ Je souhaite recevoir les offres promotionnelles des partenaires des Éditions Diamond.

En envoyant ce bon de commande, je reconnais avoir pris connaissance des conditions générales de vente des Éditions Diamond à l'adresse internet suivante : boutique.ed-diamond.com/content/3-conditions-generales-de-ventes et reconnais que ces conditions de vente me sont opposables.

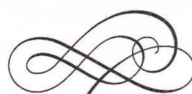
HACKABLE MAGAZINE

Les Éditions Diamond
Service des Abonnements
10, Place de la Cathédrale
68000 Colmar – France
Tél. : + 33 (0) 3 67 10 00 20
Fax : + 33 (0) 3 67 10 00 21

Vos remarques :

QUELLES APPLICATIONS ANDROID UTILISER POUR EXPLORER RFID ET NFC ?

Denis Bodor



Google Play est plein à craquer d'applications de toutes sortes, des bonnes, des passables, des médiocres, des pathétiques, et des littéralement douteuses demandant des autorisations bien étranges... Pour trouver les « bonnes » applications, souples, puissantes, stables et bien conçues, il n'y a pas d'autre choix que le test systématique. Une chance pour vous, j'ai réalisé cette démarche et vous livre ici la sélection d'outils que j'utilise régulièrement.

Le système d'évaluation et de commentaire sur Google Play peut tantôt être utile, mais uniquement à la condition que l'ensemble des utilisateurs fasse un usage, même approximatif, de la masse gélatineuse logée dans leur boîte crânienne. En d'autres termes, il n'est pas rare qu'un sombre idiot « descende » une application dont il ne connaît ni le fonctionnement ni le propos. Lorsqu'il s'agit d'un jeu populaire avec des centaines de milliers d'installations, l'évaluation fournie par l'idiot en question est naturellement pondérée. Avec des applications « techniques » ciblant un public réduit, c'est une tout autre affaire et les appréciations illégitimes prennent une proportion non négligeable. De plus, dans le cas présent, la compatibilité des smartphones entre également en jeu et nombreux sont ceux qui préfèrent accuser l'application que leur téléphone adoré.

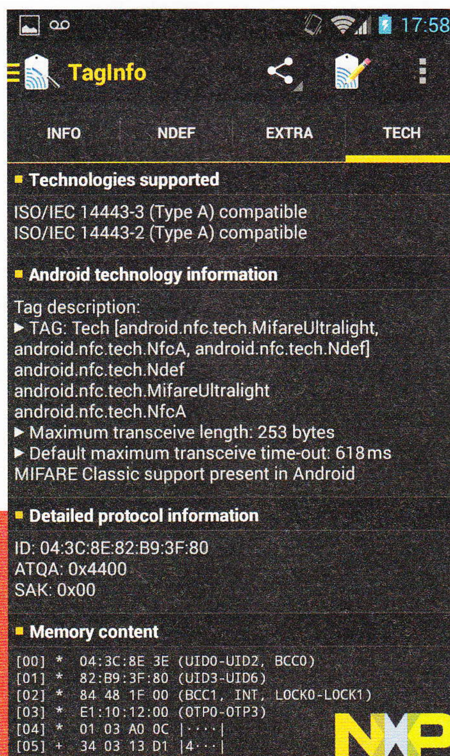
Rien de mieux donc que de faire l'expérience, installer toutes les applications de près ou de loin en rapport avec les technologies NFC ou RFID, les tester, se documenter sur leurs fonctionnalités, leurs histoires et leurs origines. Chose qui, au fil du temps, se fait naturellement et qui découle sur une sélection réduite d'applications pour un usage presque quotidien.

Les quelques applications qui suivent sont donc celles ayant obtenu

mes faveurs et mes préférences. Cette liste n'est ni objective ni exhaustive. D'une part, il s'agit de mes préférences personnelles et d'autre part, rien ne dit que je n'ai pas omis une application fabuleuse, gratuite, ergonomique et combinant les fonctionnalités de toutes les autres (c'est peu probable cependant).

Précisons également que ces applications nécessitent un smartphone sous Android compatible NFC (et plus si affinités). Comprenez par là, comme précisé dans l'article d'introduction, que tous les smartphones ne sont pas égaux. Ceux équipés de puces NXP sont capables de lire et écrire les tags NXP Mifare Classic (S50 et S70), les autres se limitent aux standards strictement définis par la norme NFC. Avant de blâmer une application, assurez-vous de la compatibilité de votre appareil. La sélection qui suit est celle présente sur mon Samsung Galaxy Nexus sous Android 4.3 (dernière version disponible pour ce modèle).

1 NFC TAGINFO BY NXP



Celle-ci se limite à la lecture des tags, mais avec un niveau de technicité très important. L'application vous fournira ainsi des indications sur le type de tag détecté et lu, son origine, son modèle exact, la nature des données s'y trouvant (NDEF ou non), les données elles-mêmes (dans la mesure du possible en fonction des restrictions de sécurité configurées), etc.

Le design de l'application est très agréable et il est possible d'ajuster les préférences de façon à utiliser des clés et mots de passe personnalisés pour Mifare Classic, Ultralight C, Ultralight EV1 et NTAG21x. Ceci est cependant limité dans la configuration de façon à éviter de rendre un tag inutilisable par inadvertance. Les NTAG avec authentification, par exemple, peuvent utiliser un compteur limitant le nombre de tentatives d'accès. Cette application ne tentera pas l'authentification si le paramètre en question est activé (*AUTHLIM*) ou si la configuration du tag est illisible (et donc la vérification de la présence de *AUTHLIM* impossible).

L'application cependant n'est pas parfaite et la présentation des restrictions de sécurité pour certains tags mériterait d'être éclaircie. L'application suivante est bien plus efficace à ce niveau. Il en va de même pour la présentation des données NDEF qui bien que lisible reste ici relativement spartiate.

Voilà sans doute l'application classique incontournable puisqu'elle est développée et proposée par *NXP Semiconductors* qui je le rappelle est l'un des fondateurs du NFC Forum et un des principaux fabricants de tags et de périphériques RFID et NFC.

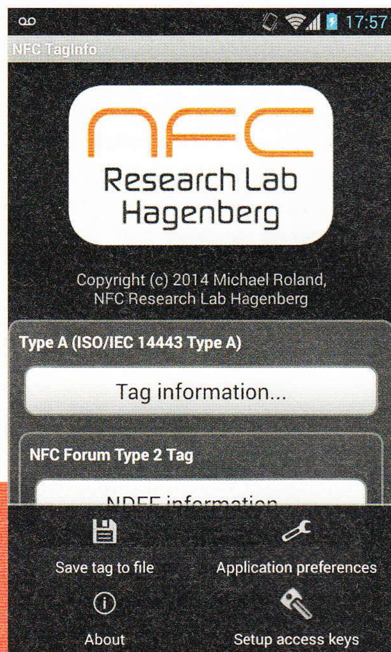


PRIX : Gratuit

COMPATIBILITÉ : Android 2.3.3 ou version ultérieure

LIEN : https://play.google.com/store/apps/details?id=com.nxp.taginfolite

2 NFC TAGINFO



Cette application proposée par le *NFC Research Lab Hagenberg* se limite également à la lecture de tags, mais avec un certain nombre de différences. L'apparence de l'application est un peu moins « moderne » que celle de NXP mais, sur certains points, bien plus agréable.

À la lecture d'un tag, une présentation sommaire liste une série de boutons permettant d'accéder aux informations souhaitées comme les informations sur le tag lui-même (modèle, type, constructeur, etc.), les données NDEF si présentes, une déclinaison brute des données stockées (taille, hexa, ASCII, UTF-8) et aussi et surtout les restrictions d'accès admirablement présentées.

En dehors des deux points forts que sont la présentation des données NDEF et des conditions d'accès, cette application brille par sa capacité à lire un grand nombre de types de tags NFC ou non (RFID 13,56 Mhz). Ceci incluant des tags qu'il ne m'a pas été possible de lire avec l'application NXP à un moment donné (les mises à jour sont très fréquentes sur les applications NXP).

Par habitude, lorsque je tombe sur un nouveau tag ou lors de la réception d'une commande, j'utilise généralement cette application de concert avec celle de NXP, ne serait-ce que pour confirmer la cohérence des informations obtenues de part et d'autre.

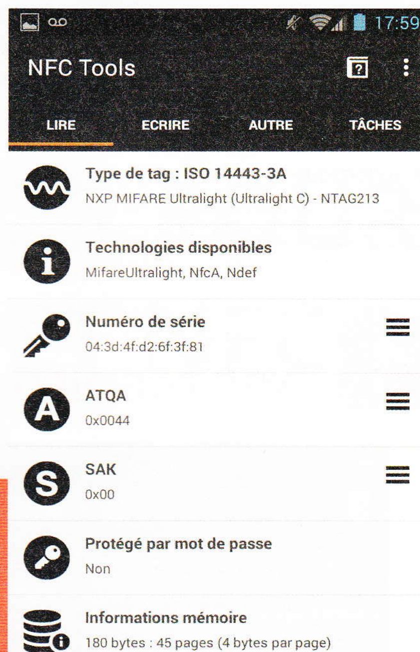


PRIX : Gratuit

COMPATIBILITÉ :
Android 2.3.3 ou version ultérieure

LIEN : https://play.google.com/store/apps/details?id=at.mroland.android.apps.nfctaginfo

3 NFC TOOLS



Voilà une application qui fait plaisir à voir ! Non seulement elle titille mon côté chauvin puisqu'elle est développée par un français, Julien Veuillet (alias *wakdev*), mais en plus elle combine agréablement l'aspect technique et la partie fonctionnelle.

Elle permet la lecture en fournissant des informations précises comme le type, modèle et ID de la carte... mais offre également la possibilité d'enregistrer des données NDEF sur un tag comme le fait le *NFC TagWriter* de NXP. L'ensemble est cependant bien plus agréable et convivial, et permettra de rapidement découvrir l'étendue des possibilités offertes par les technologies NFC sur smartphones.

À choisir entre *NFC TagWriter* et cette application, ma préférence penche nettement en faveur de cette dernière, même si un certain nombre de fonctionnalités manquent à l'appel dans la version gratuite. En effet,

alors que *NFC TagWriter* se base nativement sur un mécanisme de *datasets*, *NFC Tools* inclus ce type de choses sous la forme d'une gestion de profils uniquement disponible dans l'édition Pro, disponible pour quelques malheureux 2,99€.

À ce propos d'ailleurs, il est agréable de constater que certains développeurs d'applications Android ont la bonne idée de ne pas proposer d'applications gratuites trop restrictives dans le simple but de pousser à l'achat d'une version Pro ou Deluxe. Bref, de ne pas faire resurgir le vieux concept de *shareware* d'outre-tombe où il se trouve très bien.

NFC Tools m'a entièrement satisfait sans la gestion de profils de l'édition Pro, mais pour autant, je ne me suis pas retenu de l'acheter, ne serait-ce que pour gratifier le développeur de mon appréciation selon le bon vieux principe « tout travail mérite salaire ». En réalité, je pense avoir davantage dépensé 3€ pour la version que j'avais déjà à ma disposition gratuitement plutôt que pour la version Pro.

Notez qu'il existe également une application additionnelle nommée *NFC Tasks* permettant d'exécuter des tâches NFC automatiquement. *NFC Tools* permet en effet d'enregistrer sur un tag des actions sous forme de messages NDEF, comme activer/désactiver le Wifi, passer en « mode avion », définir un niveau de volume audio, etc. En installant *NFC Tasks* en plus de *NFC Tools*, les actions enregistrées seront alors exécutées dès la lecture du tag en question. *NFC Tasks* est également gratuit et ne nécessite pas l'édition Pro de *NFC Tools*.



PRIX : Gratuit

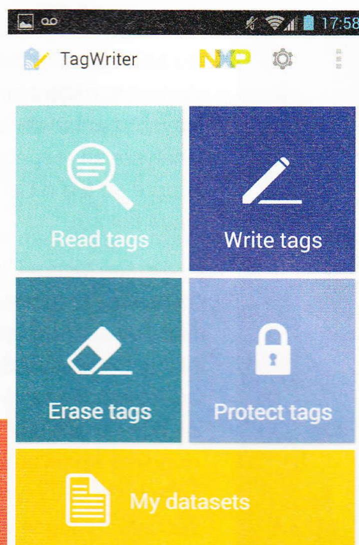
PRIX ÉDITION PRO : 2,99€

COMPATIBILITÉ : Android 4.0 ou version ultérieure

LIEN : https://play.google.com/store/apps/details?id=com.wakdev.wdnfc

LIEN ÉDITION PRO : https://play.google.com/store/apps/details?id=com.wakdev.nfctools.pro

4 NFC TAGWRITER BY NXP



Cette application de chez *NXP Semiconductors* est la grande sœur de *NFC TagInfo* et propose aussi bien une fonction de lecture que d'écriture NDEF. Celle-ci est davantage orientée « utilisation courante » (ou *enduser*) en proposant de lire et d'écrire différents types de messages NDEF dans un tag : lien, coordonnées Wifi, e-mail, SMS, position géographique, lancement d'application, etc.

La gestion de ces messages prend la forme de *datasets* qu'on peut voir comme un catalogue de messages NDEF regroupant aussi bien ceux que vous composez dans l'application que ceux lus depuis des tags.

Notez au passage que l'intérêt principal de cette application concerne le NFC au sens strict du terme et donc uniquement des données NDEF avec des tags capables de stocker ce type d'informations. Des tags lisibles avec un smartphone NFC, comme le type V (Vincity) Tag-it de Ti, ne peuvent donc être utilisés avec cette application.

L'usage primaire de cette application consistera au stockage NDEF aussi bien pour une utilisation classique (*smart poster*, transmission de coordonnées, etc.) que pour les essais de lecture via d'autres biais (LibNFC sur Raspberry Pi, module PN532 avec Arduino, etc.). L'idée est donc ici de se composer des données NDEF et voir si l'on peut les lire par ailleurs.

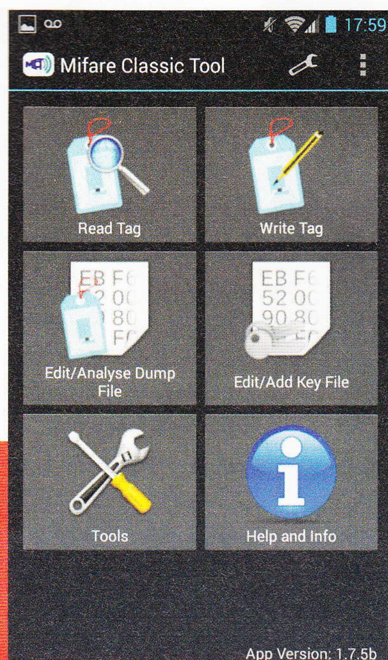


PRIX : Gratuit

COMPATIBILITÉ : Android 4.1 ou version ultérieure

LIEN : https://play.google.com/store/apps/details?id=com.nxp.nfc.tagwriter

5 MIFARE CLASSIC TOOL - MCT



Voici une autre application que j'apprécie beaucoup, issue d'un développeur créatif et indépendant. En revanche, il ne s'agit pas à proprement parler d'une application NFC, mais plus exactement d'un outil de lecture/écriture de tags Mifare Classic. ATTENTION, si votre smartphone n'est pas équipé d'une puce NFC de chez NXP, cette application ne vous servira à rien, car elle ne sait manipuler que les Mifare Classic alors que votre périphérique lui sera bien incapable d'obtenir autre chose que l'UID d'un tel tag.

Mais si vous disposez d'un smartphone adapté, cette application est, selon moi, tout simplement le meilleur outil de lecture, écriture, analyse, clonage de tag Mifare Classic. Malgré une interface qui peut paraître un peu sommaire, *Mifare Classic Tool*, ou MCT pour les intimes, dispose de tout le nécessaire pour ce travail et présente les données à analyser en respectant la structure propre au tag (et le tout agréablement coloré).

Un tag Mifare Classic de 1k ou 4k est organisé en une série de secteurs de plusieurs blocs. Chaque secteur peut être configuré de manière à permettre ou non l'accès à ses blocs de données après authentification à l'aide de deux clés (A et B). La sécurité n'est pas absolue comme l'ont démontré un certain nombre de travaux de recherche au cours des dernières années. Les tags Mifare Classic produits récemment corrigent une partie des problèmes de sécurité, en particulier lorsque tous les secteurs d'un tag nécessitent une authentification.

Les Mifare classic sont largement utilisés pour bon nombre d'applications courantes allant des cartes de fidélité aux moyens de paiement à points (type laverie automatique) en passant par le contrôle d'accès. MCT sera un compagnon de choix sachant qu'il permet de configurer et stocker les clés et donc d'accéder aux tags n'utilisant pas les clés par défaut (`ffffffffffff`, `a0b0c0d0e0f0`, etc.).

De plus, et c'est là sans doute le point le plus important, cette application est un logiciel libre sous licence GNU GPL et ses sources sont disponibles sur GitHub (**https://github.com/ikarus23/MifareClassicTool**). Son créateur, Gerhard Klostermeier, a choisi cette ouverture de façon non seulement à proposer un outil fiable, mais également à permettre les contributions externes (pour en savoir plus sur ses failles, cherchez « mfoc » et « mfcuk » sur le Web).

Notez qu'il existe deux versions, fonctionnellement identiques, de cette application sur Google Play, l'une gratuite et l'autre au prix de 2,50€, permettant ainsi de faire une forme de don au développeur et à l'aider dans le développement. Chose que je me suis, bien entendu, empressé de faire, car les applications open source de qualité ainsi disponibles sont bien trop rares.



PRIX : Gratuit

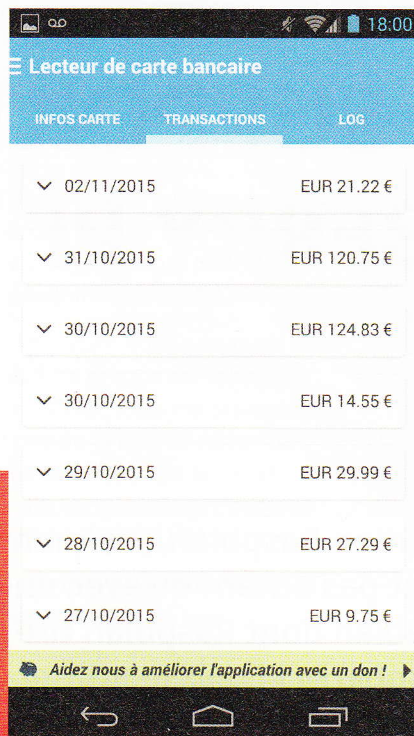
PRIX VERSION DONATE : 2,50€

COMPATIBILITÉ : Android 2.3.3 ou version ultérieure

LIEN : https://play.google.com/store/apps/details?id=de.syss.MifareClassicTool

LIEN VERSION DONATE : https://play.google.com/store/apps/details?id=de.syss.MifareClassicToolDonate

6 LECTEUR DE CARTE BANCAIRE NFC



Pour terminer, voici une application qui vous permettra surtout de vous faire peur ou de faire peur à vos amis en leur démontrant que leur carte bancaire avec paiement sans contact est très bavarde.

Ceci n'est pas une nouveauté et dès 2011 un certain nombre de spécialistes en sécurité ont tiré la sonnette d'alarme, non que le paiement sans contact soit peu sûr, c'est plutôt au niveau du respect de la vie privée que le problème se pose. Disons-le clairement, il n'est pas possible de débiter les comptes des gens en se frottant à eux dans les transports en commun (sauf s'ils vous paient pour ça (?!)), mais les premiers exemplaires en distribution des cartes EMV NFC donnaient, sans authentification et sans chiffrement, des informations comme le nom et prénom du porteur, le numéro de la carte, la liste des dernières opérations, etc.

Rien de bien extraordinaire me direz-vous puisque ces informations sont également disponibles sur la piste magnétique, dans la puce intégrée et tout simplement embossées sur la carte elle-même. La grosse différence cependant était surtout que ces informations puissent être obtenues à l'insu du propriétaire de la carte.

Depuis fin 2013, les nouvelles cartes bancaires produites avec la fonctionnalité de paiement sans contact ne divulguent plus ces informations personnelles, mais uniquement le numéro de la carte et la date d'expiration.

Cette application de Julien Millau est surtout une démonstration pour sa bibliothèque Java accomplissant le travail de décodage de l'information délivrée par les cartes. L'application comme la bibliothèque sont disponibles en logiciel libre sous licence Apache 2.0 sur GitHub : **https://github.com/devnied/EMV-NFC-Paycard-Enrollment**.

Elle vous permettra de tester rapidement les informations que votre carte bancaire divulgue qu'il s'agisse de son numéro, vos coordonnées ou les dernières transactions effectuées.

Notez que l'application disponible via Google Play est gratuite, mais qu'il est possible de faire un don au développeur pour l'aider à maintenir son code et améliorer l'application. **DB**



PRIX : Gratuit

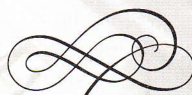
DON POSSIBLE IN APP : 2,99€, 4,99€, 9,99€, 15,66€, 23,99€ ou 36€

COMPATIBILITÉ : Android 4.0.3 ou version ultérieure

LIEN : https://play.google.com/store/apps/details?id=com.github.devnied.emvnfccard

CONFIGURER PROPREMENT LE SUPPORT NFC SUR RASPBERRY PI

Denis Bodor



Il y a beaucoup de choses dans une distribution Raspbian, vraiment beaucoup de choses. Mais il n'y a pas tout et pas autant qu'avec une distribution GNU/Linux sur PC, comme Debian dont Raspbian tire ses racines. Pour que le système de la Raspberry Pi puisse utiliser un périphérique de lecture/écriture NFC/RFID, il faut lui ajouter des logiciels. On peut le faire comme un barbare ou proprement.





En toute sincérité, je pensais que le nécessaire en termes de logiciels et bibliothèques serait immédiatement disponible dans Raspbian. Ce n'est pas le cas et pour pouvoir utiliser des tags NFC/RFID, il faudra avant tout faire un petit travail d'installation et de configuration.

La littérature que l'on trouve sur le Web à ce propos est relativement explicite, mais malheureusement pas de la qualité adéquate pour un utilisateur qui tente de respecter au mieux la philosophie du système et de la distribution.

Le site très populaire d'Adafruit par exemple, détaille une procédure pour accompagner son produit « *PN532 NFC/RFID controller breakout board - v1.6* » qui ne me convient pas du tout. Ce périphérique construit autour du contrôleur NFC PN532 de chez MXP (ex-Philips) peut être utilisé comme shield Arduino ou directement par la Raspberry Pi via son port série (en désactivant la console). Cette carte à base de PN532 n'est pas la seule solution disponible et si l'on vise une utilisation exclusive sur Pi et PC/Mac, on préférera sans doute opter pour un périphérique USB.

Le plus populaire dans cette catégorie de périphérique est sans le moindre doute le lecteur ACS ACR122U disponible pour quelques 30€ un peu partout sur le Web (et sur eBay bien entendu). C'est le plus populaire, mais pas le plus fiable et celui-ci comporte un certain nombre de bugs dans son firmware qui tantôt oblige à le déconnecter et le reconnecter pour le réinitialiser. Si on peut se permettre une dépense un peu plus importante, on préférera choisir un produit reconnu de qualité comme une clé *Identive Infrastructure* (anciennement SCM) de modèle SCL3711 à quelques 20€ de plus.



Les périphériques qui nous intéressent ici sont principalement USB avec à gauche l'ACR122U et à droite le SCL3711, mais la LibNFC est en mesure de prendre en charge d'autres types de connexions comme i2c et SPI.

Dans tous les cas, qu'il s'agisse d'un shield PN532, d'un ACR122U ou d'une clé SCM, une bibliothèque et ses outils sont l'élément indispensable : la LibNFC et les NFC-tools, qui malheureusement ne sont pas (ou pas encore) disponibles pour Raspbian. C'est le point précis adressé par la page *learn* d'Adafruit (et en particulier par Kevin Townsend), ainsi après téléchargement des sources :

```
$ tar -xvzf libnfc-1.7.0.tar.gz
$ cd libnfc-libnfc-1.7.0
$ autoreconf -vis
$ ./configure --with-drivers=pn532_uart
--sysconfdir=/etc --prefix=/usr
$ sudo make clean
$ sudo make install all
```

Clairement, ceci va saccager votre distribution Raspbian en installant, littéralement à la sauvage, des éléments sans aucune intervention du système de gestion de paquets... Bref, c'est ragoutant, ignoble,

horripilant, écœurant, indigeste et... ça me fait penser au céleri (qui possède exactement les mêmes propriétés) et provoque généralement une effusion sonore du type « *Bweurk, mais c'est dégueu* » propre à inquiéter mes collègues.

J'apprécie Adafruit, leurs efforts de démocratisation de l'électronique hobbyiste et du logiciel libre, ainsi que beaucoup de leurs produits, mais ce type d'explications fait, à mon goût, plus de mal que de bien sur le long terme. Cette suite de manipulations était courante il y a 15 ans (et encore), mais nous n'en sommes plus à gratter deux cailloux pour faire du feu... À jouer à cela, on se retrouve inéluctablement avec un système rafistolé et impossible à administrer ou mettre à jour. Il existe une façon de faire propre, moderne et respectueuse de la distribution.

1. SOYONS PROPRES ET BIEN ÉLEVÉS

Pour pouvoir prendre en charge un « lecteur » NFC/RFID sur un ordinateur sous GNU/Linux comme la Raspberry Pi et sa Raspbian, nous devons utiliser la LibNFC et accessoirement la libfreefare (dédiée aux tags de la famille NXP Mifare), mais aussi et surtout les outils qui sont livrés avec. Ces éléments ne sont pas dans les dépôts Raspbian, mais ils le sont dans Debian. La solution consiste donc à utiliser les seconds dans le premier. Il n'est bien entendu pas possible de prendre et d'installer les paquets binaires, mais on peut passer par les sources et générer de nouveaux paquets pour notre Raspbian.

Il y a plusieurs façons de faire, mais celle qui m'est venue immédiatement à l'esprit consiste à tout simplement récupérer les paquets sources sur un PC Debian et de transférer cela sur la Pi pour construction. Ainsi, côté PC, on se place dans un répertoire temporaire vide quelconque et on se plie d'un :

```
% apt-get source libnfc
% apt-get source libfreefare0
```

`apt-get source` récupère et prépare les sources automatiquement sur la machine. Mais les répertoires créés ne nous intéressent pas. Les éléments des paquets sources en revanche seront transférés sur la Raspberry Pi (là aussi dans un répertoire temporaire vide) :

- pour la LibNFC ce sont les fichiers `libnfc_1.7.1-4.debian.tar.xz`, `libnfc_1.7.1-4.dsc` et `libnfc_1.7.1.orig.tar.gz`,
- et pour la libfreefare : `libfreefare_0.4.0-2.debian.tar.xz`, `libfreefare_0.4.0-2.dsc` et `libfreefare_0.4.0.orig.tar.gz`.

Les fichiers `.orig.tar.gz` sont les archives des sources originales produites par les développeurs du projet NFC-Tools, les `.dsc` contiennent les descriptions des paquets sources et les `.debian.tar.xz` regroupent les éléments ajoutés et modifiés par les développeurs Debian.

On peut ensuite basculer sur la Pi et sa Raspbian afin de recréer l'arborescence permettant la construction. Cette méthode évite de copier toute une flopée de fichiers et nous permet également de voir comment désarchiver des sources Debian/Raspbian à la fois avec `apt-get` et `dpkg-source` :

```
% dpkg-source -x libnfc_1.7.1-4.dsc
% dpkg-source -x libfreefare_0.4.0-2.dsc
```

Ces deux commandes génèrent les répertoires `libnfc-1.7.1` et `libfreefare-0.4.0`. Il ne reste plus qu'à lancer la construction/compilation avec :

```
% cd libnfc-1.7.1/
% dpkg-buildpackage -b
```

À ce stade, il est possible qu'une erreur survienne, l'outil vous signifiant qu'il manque des paquets (dépendances de

construction) pour procéder à l'opération (chez moi `dh-autoreconf` et `libusb-dev`). Ceci est totalement dépendant de ce que vous avez installé précédemment sur votre Raspbian. Ajoutez simplement les paquets demandés avec `sudo apt-get install` et relancez la construction :

```
% dpkg-buildpackage -b
dpkg-buildpackage: paquet source libnfc
dpkg-buildpackage: version source 1.7.1-4
[...]
Selected drivers:
   acr122_pcsc...... no
   acr122_usb....... yes
   acr122s.......... yes
   arygon........... yes
   pn53x_usb........ yes
   pn532_uart....... yes
   pn532_spi....... yes
   pn532_i2c........ yes
[...]
```

Normalement, vous devriez configurer les sources afin, entre autres choses, de choisir quel matériel vous souhaitez que la LibNFC supporte. Dans la documentation Adafruit en l'occurrence, ceci se limite au PN532 sur port série (puisque c'est le produit qu'ils vendent), mais dans les sources Debian on peut voir que presque tout le matériel sera pris en charge, aussi bien le PN532 en série, SPI ou i2c, mais également les lecteurs USB compatibles comme l'ACR122U et les clés USB à base de composants de la famille PN53* (le SCL3711 utilise un PN533). Seul ACR122 via PC/SC Lite n'est pas activé. Sans entrer dans le détail, PC/SC (pour *Personal computer/Smart Card*) est un *middleware* s'intercalant entre le matériel et vos applications, permettant de factoriser les méthodes d'accès aux lecteurs de smartcards compatibles CCID/ICCD. Nous n'en avons pas l'usage ici et ceci nous obligerait, en plus, à installer PC/SC Lite (qui est aussi très amusant pour lire les cartes à puce, mais ce sera pour une autre fois).

Après la construction, nous obtenons, dans le répertoire parent, les paquets tant attendus :

```
% cd ..
% ls *.deb
libnfc5_1.7.1-4_armhf.deb
libnfc5-dbg_1.7.1-4_armhf.deb
libnfc-bin_1.7.1-4_armhf.deb
libnfc-dev_1.7.1-4_armhf.deb
libnfc-examples_1.7.1-4_armhf.deb
libnfc-pn53x-examples_1.7.1-4_armhf.deb
```

Il nous suffit alors de les installer avec :

```
% sudo dpkg -i *.deb
[...]
Paramétrage de libnfc5:armhf (1.7.1-4) ...
Paramétrage de libnfc5-dbg:armhf (1.7.1-4) ...
Paramétrage de libnfc-bin (1.7.1-4) ...
Paramétrage de libnfc-dev:armhf (1.7.1-4) ...
Paramétrage de libnfc-examples (1.7.1-4) ...
Paramétrage de libnfc-pn53x-examples (1.7.1-4) ...
Traitement des actions différées (" triggers ") pour " man-db "...
```

Et maintenant que nous sommes bien échauffés, nous pouvons récidiver avec la libfreefare :

```
% cd libfreefare-0.4.0/

% dpkg-buildpackage -b

% cd ..

% sudo dpkg -i  libfreefare*.deb
[...]
Paramétrage de libfreefare0:armhf (0.4.0-2) ...
Paramétrage de libfreefare-bin (0.4.0-2) ...
Paramétrage de libfreefare-dev:armhf (0.4.0-2) ...
Paramétrage de libfreefare-doc (0.4.0-2) ...
Traitement des actions différées (" triggers ") pour " man-db "...
```

Et voilà, nous avons maintenant tout le nécessaire pour prendre en charge notre matériel et disposons de nombreux outils pour lire et écrire des tags RFID ou NFC de tous types.

2. UNE POINTE DE CONFIGURATION AVANT DE BRANCHER

Installer les éléments logiciels nécessaires est une chose, faire en sorte que le système se comporte de façon sympathique avec nous en est une autre. Cela tient en deux points : dire au noyau de ne pas ajouter son grain de sel et permettre à un utilisateur standard d'utiliser le matériel (sans avoir à utiliser `sudo` donc).

La première étape consiste à dire au noyau de ne jamais charger les modules permettant la prise en charge de certains lecteurs RFID/NFC (exactement comme nous l'avions fait dans un précédent numéro pour le récepteur DVB-T). On créé donc un fichier `/etc/modprobe.d/nfc-blacklist.conf` contenant les noms des modules à placer sur la liste noire :

```
blacklist pn533
blacklist nfc
```

Il seront dès lors interdits de chargement et ne perturberons pas les applications basées sur la LibNFC, qui elle-même utilise la LibUSB pour accéder directement au matériel.

On se tourne ensuite vers Udev pour lui demander de changer automatiquement les permissions sur le périphérique lors de la connexion/détection. Nous ajoutons donc un fichier `/etc/udev/rules.d/nfc.rules` contenant :

```
# ACR122 / Touchatag
SUBSYSTEM=="usb", ACTION=="add", ATTRS{idVendor}=="072f",\
ATTRS{idProduct}=="2200", MODE="0660", OWNER="root", GROUP="plugdev"
# SCL SCL3711
SUBSYSTEM=="usb", ACTION=="add", ATTRS{idVendor}=="04e6",\
ATTRS{idProduct}=="5591", MODE="0660", OWNER="root", GROUP="plugdev"
```

Ces lignes concernent respectivement les périphériques USB identifiés par `072f:2200` et `04e6:5591`, ce qui correspond aux *vendorID:productID* de l'ACR122U et du SCL3711. Il suffira ensuite de relancer le service adéquat avec :

```
% sudo service udev restart
[ ok ] Stopping the hotplug events dispatcher: udevd.
[ ok ] Starting the hotplug events dispatcher: udevd.
```

3. EST-CE QUE ÇA MARCHE ?

Il nous suffit maintenant de brancher un lecteur USB, ici la clé SCM Micro SCL3711, et de demander à `nfc-list`, un outil livré avec la LibNFC :

```
$ nfc-list
nfc-list uses libnfc 1.7.1
NFC device: SCM Micro / SCL3711-NFC&RW opened
```

Le lecteur est parfaitement reconnu sans avoir à utiliser `sudo`. Nous pouvons alors tenter la lecture d'un tag avec :

```
$ nfc-poll
nfc-poll uses libnfc 1.7.1
NFC reader: SCM Micro / SCL3711-NFC&RW opened
NFC device will poll during 30000 ms
  (20 pollings of 300 ms for 5 modulations)
ISO/IEC 14443A (106 kbps) target:
    ATQA (SENS_RES): 00  04
       UID (NFCID1): 02  7d  2b  ca
      SAK (SEL_RES): 08
nfc_initiator_target_is_present: Target Released
Waiting for card removing...done.
```

On peut également utiliser les outils de la libfreefare pour obtenir des informations sur un tag Mifare Ultralight par exemple (ici avec l'ACR122U) :

```
$ mifare-ultralight-info
Tag with UID 04b25322722f80 is a Mifare UltraLight
```

Ou encore faire de même avec un tag Mifare DESFire EV1 :

```
$ mifare-desfire-info
===> Version information for tag 0456595afc2e80:
UID:                        0x0456595afc2e80
Batch number:               0xba4415aba0
Production date:            week 5, 2013
Hardware Information:
    Vendor ID:              0x04
    Type:                   0x01
    Subtype:                0x01
    Version:                1.0
    Storage size:           0x1a (=8192 bytes)
    Protocol:               0x05
Software Information:
    Vendor ID:              0x04
    Type:                   0x01
    Subtype:                0x01
```

```
    Version:              1.4
    Storage size:         0x1a (=8192 bytes)
    Protocol:             0x05
Master Key settings (0x0f):
    0x08 configuration changeable;
    0x04 PICC Master Key not required for create / delete;
    0x02 Free directory list access without PICC Master Key;
    0x01 Allow changing the Master Key;
Master Key version: 0 (0x00)
Free memory: 64 bytes
Use random UID: no
```



Et voici une Raspberry Pi prête pour vous faire découvrir le monde extraordinaire du NFC grâce aux nombreuses bibliothèques en logiciel libre et aux outils qui les accompagnent...

Si vous comptez utiliser le produit Adafruit, rien de plus simple, il suffit d'éditer le fichier **/etc/nfc/libnfc.conf**, installé en même temps que les paquets et de préciser quelques lignes, comme indiqué dans la documentation Adafruit :

```
allow_intrusive_scan = false
device.name = "PN532 board UART"
device.connstring = "pn532_uart:/dev/ttyAMA0"
```

Vous voici donc maintenant propriétaire d'un système capable de lire et d'écrire des tags RFID/NFC et ce, tout en gardant un système bien propre et cohérent. Si l'envie vous passe, vous pourrez supprimer ces éléments à l'aide d'un simple **sudo apt-get remove**, chose littéralement impossible avec la méthode barbare d'il y a 15 ans...

4. PETIT BONUS : LA LIBNDEF ET SES OUTILS

Un autre élément géré par le projet NFC-Tools s'appelle libNDEF qui, comme son nom l'indique, est une bibliothèque permettant de faciliter le décodage et l'encodage de données au format NDEF. Ici, la technique consiste à également produire un paquet, mais à partir des sources récupérées sur GitHub :

```
% mkdir kkpart
% cd kkpart
% git clone https://github.com/nfc-tools/libndef.git
% cd libndef/
```

Cette bibliothèque est livrée avec deux programmes exemples très intéressants, mais qui ne font pas parti du contenu du paquet par défaut. Comme c'est là précisément ce qui nous intéresse, nous modifions le paquet. Ainsi, nous ajoutons dans le fichier `debian/libndef.dirs` une ligne contenant :

```
usr/bin
```

Et dans le fichier `debian/libndef.install`, deux lignes précisant :

```
usr/bin/ndef-decode
usr/bin/ndef-encode
```

On peut ensuite enchaîner sur la construction :

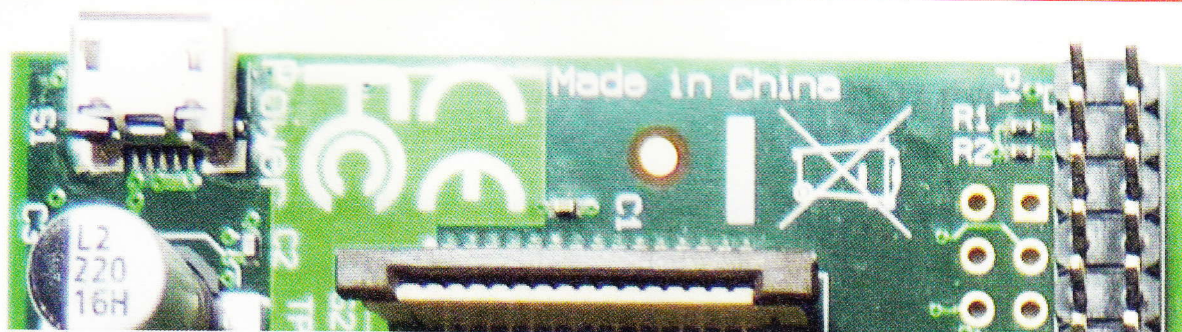
```
% dpkg-buildpackage -b
[éventuellement installer les paquets demandés]
[chez moi libqt4-dev et qt4-qmake]
% dpkg-buildpackage -b
% sudo dpkg -i ../libndef*.deb
```

Notre modification nous permet désormais de disposer des outils `ndef-decode` et `ndef-encode` pour respectivement décoder et encoder du NDEF. Si l'on utilise ces outils de concert avec ceux des autres paquets nous pouvons lire directement les informations d'un tag NFC contenant des données formatées :

```
% mifare-classic-read-ndef -y -o - | ndef-decode
Use stdin as input file
NFC Forum application contains a "NULL TLV", Skipping...
NFC Forum application contains a "NULL TLV", Skipping...
NFC Forum application contains a "NDEF Message TLV".
NDEF message is valid and contains 2 NDEF record(s).
NDEF record (1) type name format: NFC Forum well-known type
NDEF record (1) type: T
NDEF record (1) payload (language): English (en)
NDEF record (1) payload (text): coucou plop
NDEF record (2) type name format: NFC Forum external type
NDEF record (2) type: android.com:pkg
NDEF record (2) payload (hex): 6e65742e626974686561702e736964706c61796572
```

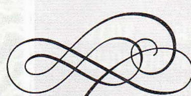
Ce tag enregistré avec un smartphone Android contient un message NDEF avec deux enregistrements, respectivement un simple texte et des données de type « paquet Android » permettant de lancer une application (`net.bitheap.sidplayer` pour SIDplayer, mais ceci est affiché en valeurs hexadécimales).

Ce résultat est bien entendu perfectible puisqu'il ne s'agit d'exemples d'utilisation de la bibliothèque. Dommage que ce soit du C++ reposant sur Qt, je me serai potentiellement lancé dans l'aventure... DB

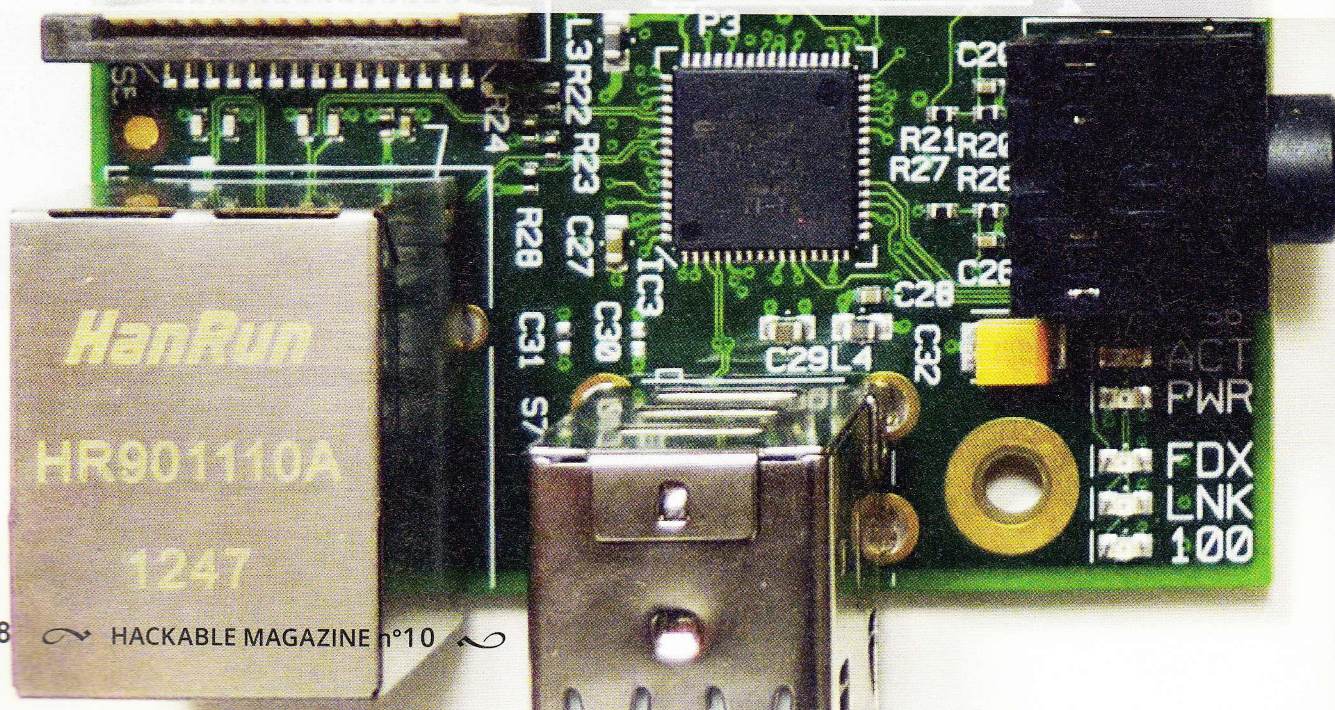


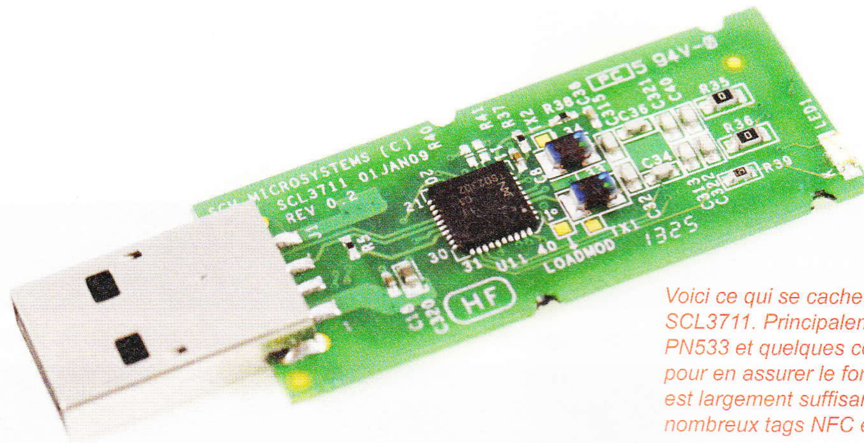
S'AMUSER AVEC LES TAGS RFID/NFC SUR UNE RASPBERRY PI

Denis Bodor



Dans un article précédent, nous avons vu comment proprement installer les bibliothèques et les outils permettant de prendre en charge un lecteur de tags RFID et NFC. Il est temps à présent de faire un petit tour des utilitaires à notre disposition et de ce qu'ils permettent de faire.





Voici ce qui se cache dans un lecteur SCM SCL3711. Principalement une puce NXP PN533 et quelques composants passifs pour en assurer le fonctionnement. Ceci est largement suffisant pour lire et écrire de nombreux tags NFC et RFID.

Les LibNFC et libfreefare que nous avons installés sont livrés avec un certain nombre d'utilitaires. Ceux-ci sont davantage des exemples d'utilisation des bibliothèques que des outils à part entière. Il n'en reste pas moins cependant qu'ils sont parfaitement utilisables pour manipuler différents types de tags, y écrire et y lire des données formatées NDEF ou autres.

Nous partirons ici du principe que les manipulations décrites dans l'article précédent à propos de l'installation sur Raspberry Pi ont été menées à bien et donc que le matériel est correctement pris en charge. Si vous avez sauté directement à cet article en espérant éviter la lecture de l'introduction sur les technologies RFID et NFC, vous risquez de rencontrer des difficultés, car je ne reviendrai pas sur les acronymes et les principes généraux (c'est tout l'intérêt d'une introduction).

1. EN SAVOIR PLUS SUR UN TAG

Nous avons quelques outils à notre disposition pour obtenir des informations à la fois sur le lecteur (PCD) et sur le ou les tags (PICC). Ainsi, nous pouvons déjà vérifier que le périphérique est correctement pris en charge avec :

```
$ nfc-scan-device
nfc-scan-device uses libnfc 1.7.1
1 NFC device(s) found:
- SCM Micro / SCL3711-NFC&RW:
    pn53x_usb:001:006
```

Nous obtenons le modèle exact (SCL3711) ainsi que le type de connexion (USB). Nous pouvons alors enchaîner avec `nfc-poll` destiné à s'enquérir d'un tag sur le premier lecteur trouvé :

```
$ nfc-poll
nfc-poll uses libnfc 1.7.1
NFC reader: SCM Micro / SCL3711-NFC&RW opened
NFC device will poll during 30000 ms
(20 pollings of 300 ms for 5 modulations)
```

L'outil attend ensuite qu'un tag soit présenté puis retiré :

```
ISO/IEC 14443A (106 kbps) target:
    ATQA (SENS_RES): 00  44
       UID (NFCID1): 04  78  3d  ca  ca  2b  80
      SAK (SEL_RES): 00
nfc_initiator_target_is_present: Target Released
Waiting for card removing...done.
```

Nous voyons alors différentes informations sous la forme de valeurs obtenues directement du tag, comme son UID. Nous voyons également des valeurs pour `ATQA` et `SAK`. Il s'agit d'APDU, une désignation généralement utilisée pour les cartes à puce, ou en d'autres termes de « commandes » dans le langage de communication entre le lecteur et la carte. Les valeurs retournées permettent de déduire le type de tag présent.

En utilisant l'option `-v` de la commande, on pourra obtenir davantage d'informations, dont le type de tag en clair, comme par exemple :

```
Fingerprinting based on MIFARE type Identification Procedure:
* MIFARE Ultralight
* MIFARE Ultralight C
```

ou

```
Fingerprinting based on MIFARE type Identification Procedure:
* MIFARE Classic 1K
* MIFARE Plus (4 Byte UID or 4 Byte RID) 2K, Security level 1
* SmartMX with MIFARE 1K emulation
```

Attention, cette information n'est pas 100% fiable. Le premier exemple est celui obtenu avec un tag NTAG213 de NXP, technologiquement très proche (sinon similaire) à un tag Ultralight C. Le second exemple correspond à un tag Mifare Classic 1K, mais comme on peut le voir, la distinction avec un Mifare Plus ou un tag SmartMX émulant un Mifare Classic (courant pour certaines cartes bancaires) n'est pas possible aussi simplement.

Un autre outil, `nfc-list`, vous affichera les mêmes informations, y compris avec l'option `-v`. Les deux utilitaires se différencient par le mode d'utilisation du lecteur (polling dans le cas de `nfc-poll` et lecture directe ponctuelle avec `nfc-list`).

En cas de doute sur un modèle de tag, l'utilisation de l'application Android NXP TagInfo ou NFC Taginfo sera préférable, avec un smartphone compatible (et de préférence un contrôleur NFC NXP).

L'application Android MifareClassicTool (MCT pour les intimes) est spécialement dédiée à la manipulation des Mifare Classic et sera un compagnon de choix pour confirmer vos expérimentations. Notez la colorisation du dernier bloc de chaque secteur avec à gauche la clé A, à droite la clé B et au centre les octets déterminant la configuration du secteur.



2. LIRE ET ÉCRIRE DES MIFARE CLASSIC

Les tags Mifare Classic de NXP (ou des cloneurs chinois) sont tout simplement les plus courants, les plus faciles à trouver et souvent les moins chers. Mais ce sont aussi les tags qui ne sont pas 100% compatibles NFC et ne fonctionneront pas avec tous les lecteurs et smartphones.

Les Mifare Classic sont des tags particuliers dont la mémoire est organisée en secteurs et en blocs. 16 secteurs de 4 blocs de 16 octets pour les 1K (S50) et 40 secteurs de 4 ou 16 blocs de 16 octets pour les 4K (S70). Chaque secteur possède un bloc particulier configurant la façon de lire ou d'écrire des données dans le bloc, dépendant de deux clés secrètes (A et B). La mémoire n'est donc pas organisée de façon linéaire et tous les secteurs peuvent être configurés différemment. Ceci pose problème pour les données au format NDEF et impose l'utilisation d'une technique particulière. Le premier secteur (et le 16ème pour une 4K) ne contient pas de donnée « utilisateur », mais un répertoire appelé MAD pour *Mifare Application Directory*. Celui-ci référence quels secteurs sont utilisés pour certains types d'applications et donc quels secteurs contiennent ou non des données NDEF.

Bien sûr, vous pouvez configurer vos tags Mifare Classic comme il vous plaît, mais le standard précise l'utilisation du MAD si vous voulez utiliser ce type de tags comme compatible NFC. Le MAD, et donc le secteur 0, doit respecter un format précis et pouvoir être lisible avec une clé prédéfinie (`0xA0A1A2A3A4A5`). Les secteurs dits NDEF, eux aussi, utilisent une clé par défaut `0xD3F7D3F7D3F7` et une configuration par défaut.

Notre objectif ici est de nous en tenir aux informations au format NDEF, plus facile à manipuler. Si vous souhaitez davantage d'informations sur l'utilisation « native » des tags Mifare Classic, je vous recommande la lecture du dossier consacré à ce sujet paru dans un autre de nos magazines (*Open Silicium n°12*). Les articles (rédigés par votre serviteur) y détaillent de manière plus technique, et sans doute moins ludique, l'utilisation de ces tags sous GNU/Linux. Vous pouvez également simplement vous référer à la documentation de NXP sur le sujet, facilement accessible en ligne en cherchant « mifare classic pdf » sur le Web. L'outil à utiliser pour ce genre de choses est `nfc-mfclassic` pour lire et écrire tout ou partie d'un tag.

En nous en tenant au NDEF, nous nous facilitons la vie et n'avons pas à gérer les clés ou nous occuper du MAD. Tout se fait en coulisse, ce qui est une très bonne chose. En effet, il est possible de rendre un tag totalement inutilisable en jouant avec la configuration et oubliant/perdant les clés. Mais ceci ne sera pas un problème en nous en tenant au NDEF.

Avant toutes choses, commençons par effacer un tag avec :

```
$ mifare-classic-format
Found Mifare Classic 1k with UID 1009d74b. Format [yN] y
Formatting 16 sectors [...4...8...12...16] done.
```

Ceci a pour effet de remplir entièrement le tag de données à `0x00`, en configurant les permissions d'accès à leur valeur par défaut et en utilisant les clés par défaut `0xFFFFFFFFFFFF`. Notre tag est dès lors vierge et prêt pour y placer des données.

Il faudra utiliser `ndef-encode` pour produire des données formatées NDEF. Comme nous l'avons vu dans l'article d'introduction, ces données sont structurées et utilisent différents types (TLV). Vous n'avez pas à vous soucier de cela avec `ndef-encode` qui composera les données en fonction des options utilisées. Ainsi pour, stocker une URL comme `http://www.hackable.fr` en NDEF, il nous suffit de faire :

```
$ ndef-encode -u http://www.hackable.fr mes_donnees.ndef
```

On se retrouvera avec un fichier `mes_donnees.ndef` contenant l'adresse spécifiée. `ndef-encode` se sera chargé, comme un grand, d'encoder le fait qu'il s'agisse d'une URI et un « type bien connu », ainsi que `http://www.` en type d'enregistrement. L'ensemble occupe alors 16 malheureux octets que nous pourrons ensuite enregistrer dans le tag avec :

```
$ mifare-classic-write-ndef -i mes_donnees.ndef
NDEF file is 16 bytes long.
Found Mifare Classic 1k with UID 1009d74b. Write NDEF [yN] y
```

Dès lors, on pourra vérifier que tout cela a bien fonctionné en utilisant, par exemple, un smartphone Android et *NFC TagInfo* ou *NFC Tools*. Mais on pourra également immédiatement utiliser `mifare-classic-read-ndef` et `ndef-decode` de la même manière que les utilitaires précédents. Mais, comme `mifare-classic-read-ndef` est en mesure d'envoyer les données lues directement sur la sortie standard (console/écran), on pourra procéder à la lecture et au décodage en une seule fois :

```
$ mifare-classic-read-ndef -y -o - | ndef-decode
Use stdin as input file
Found Mifare Classic 1k with UID 1009d74b.
NFC Forum application contains a "NDEF Message TLV".
NDEF message is valid and contains 1 NDEF record(s).
NDEF record (1) type name format: NFC Forum well-known type
NDEF record (1) type: U
NDEF record (1) payload (uri): http://www.hackable.fr
```

Les options `-y` et `-o -` permettent respectivement d'éviter de devoir confirmer la lecture et spécifier le fichier en sortie, `-` représentant la sortie standard en lieu et place d'un fichier. Le `|` est ensuite utilisé pour rediriger cette sortie directement vers l'entrée de `ndef-decode` qui s'empresse de tout décoder. Celui-ci trouve un enregistrement NDEF, avec un TLV « bien connu » et un type « U » pour URI, puis nous affiche le contenu (avec le `http://www.` bien traduit).

`ndef-encode` ne se limite pas aux URI, il peut également encoder du texte (`-t`), un type MIME (`-m`) comme une carte de visite (`text/x-vCard`) à partir d'un fichier VCF, ou encore un enregistrement *SmartPoster* (titre + URI).

Une utilisation pratique de ces outils peut être celle du lecteur MP3. En effet, il vous suffira d'utiliser une URI comme `file://musique.mp3` et de l'enregistrer sur un tag comme nous venons de le faire avec `http://www.hackable.fr`. Ensuite, une simple ligne de commandes suffira : `mplayer ~/'mifare-classic-read-ndef -y -o - 2>/dev/null | ndef-decode 2>&1 | grep payload | sed "s/.*file:\/\///"'`.

« Ahhhhhh ! Il est malade ! ». Non, non, c'est plus simple que vous ne le pensez, il suffit de découper en petits morceaux :

- `mifare-classic-read-ndef -y -o - 2>/dev/null` : on utilise `mifare-classic-read-ndef` comme précédemment, mais on ajoute `2>/dev/null`. Ceci a pour effet de supprimer les messages envoyés sur la sortie d'erreur standard (stderr). Ce qui s'affiche sur votre écran est composé de ce qui est envoyé à la sortie standard stdout pour les messages « classiques » et à la sortie d'erreur standard stderr pour les erreurs, les avertissements et les informations complémentaires. `mifare-classic-read-ndef` envoie l'UID du tag, par exemple, sur stderr, mais nous n'en voulons pas. Nous ne voulons voir que les données NDEF.
- `ndef-decode 2>&1` : là encore, on joue avec les sorties, mais cette fois on ajoute à stdout ce qui arrive sur stderr. `ndef-decode` envoie en effet ses messages sur stderr, mais nous voulons les trier/filtrer, et ceci se fait via stdout.
- `grep payload` : `grep` permet de sélectionner les lignes qui nous intéressent et oublier toutes les autres, ici celle contenant le mot « payload ». `grep` fait ce travail en prenant le texte arrivant sur stdin (entrée standard) et donc depuis le stdout de la commande précédente.
- `sed "s/.*file:\/\///"` : `sed` est un éditeur de texte en ligne de commandes, il permet de manipuler le texte et, par exemple, de faire des remplacements avec `s/texte/blabla/`. Si « texte » est trouvé, il est remplacé par « blabla ». Ici, nous cherchons « n'importe quelle quantité de n'importe quel texte » (`.*`) jusqu'à `file://` et le remplaçons par rien. Comme les `/` ont une signification particulière pour `sed`, ceux du texte recherché doivent être précédés de `\`.
- Toutes les commandes sont enchaînées avec des `|` de façon à ce que la sortie standard de l'une arrive sur l'entrée standard de l'autre.
- Le tout est placé entre `'` (apostrophe inversée, obtenue avec AltGr+7) ce qui permet d'inclure le résultat de tout l'enchaînement comme s'il s'agissait d'un morceau de ligne de commandes.
- Et enfin, on fait précéder le tout de `mplayer ~/` pour composer une nouvelle ligne à exécuter.

Au fil des redirections, le texte traité sera alors :

- après le `mifare-classic-read-ndef` quelque chose comme (les données NDEV ne sont pas affichables directement) :

```
@$
Umusique.mp3
```

• après le **ndef-decode** :

```
Use stdin as input file
NDEF message is valid and contains 1 NDEF record(s).
NDEF record (1) type name format: NFC Forum well-known type
NDEF record (1) type: U
NDEF record (1) payload (uri): file://musique.mp3
```

• après le **grep** :

```
NDEF record (1) payload (uri): file://musique.mp3
```

• et finalement, après le **sed** :

```
musique.mp3
```

Si ce résultat est utilisé sur la ligne de commandes (entre `) et qu'on ajoute **mplayer**, la commande complète sera :

```
mplayer ~/musique.mp3
```

ce qui lira le fichier **musique.mp3** placé dans le répertoire personnel de l'utilisateur courant avec Mplayer.

Bien entendu, l'URI placée dans le tag pourrait aussi être **file://Album1** et avec une ligne un peu différente nous permettra de lire tous les fichiers présents dans **~/MaMusique/Album1** (**mplayer ~/MaMusique/Album1*.mp3**). Mettez cela dans un script que vous lancez via une touche, une icône ou une entrée GPIO et vous avez un lecteur audio NFC. Taguez vos albums préférés, créez les répertoires, initialisez les tags et... posez un tag sur le lecteur, validez et votre album est joué !

3. MIFARE ULTRALIGHT ET NTAG

Le cas des Mifare Ultralight et des NTAG (tag NFC type 2) est un peu problématique. Bien qu'il existe parmi les exemples un outil **nfc-mfultralight** permettant de lire et écrire ces tags, ces manipulations ne gèrent que les données brutes. Le principe est d'obtenir un fichier binaire du contenu du tag (**r** comme *read*), le modifier avec un éditeur hexadécimal par exemple et d'enregistrer (**w** comme *write*) le tout (sur le même tag ou un autre). Opérations qui donnent quelque chose comme :

```
$ nfc-mfultralight r Monfichier.mfd
NFC device: SCM Micro / SCL3711-NFC&RW opened
Found MIFARE Ultralight card with UID: 047e9cd26f3f81
Reading 16 pages |................|
Done, 16 of 16 pages readed.
Writing data to file: dump ... Done.
```

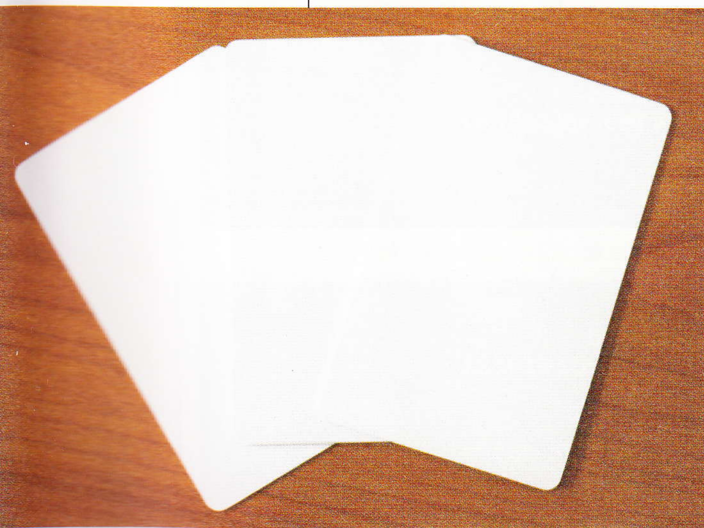
On édite des choses dans **Monfichier.mfd** et :

```
$ nfc-mfultralight w MonFichier.mfd
NFC device: SCM Micro / SCL3711-NFC&RW opened
Found MIFARE Ultralight card with UID: 047e9cd26f3f81
Write OTP bytes ? [yN] n
Write Lock bytes ? [yN] n
Write UID bytes (only for special writeable UID cards) ? [yN] n
Writing 16 pages |ssss............|
Done, 12 of 16 pages written (4 pages skipped).
```

Notez que l'outil vous demande s'il doit effectivement écrire certaines zones particulières du tag avec lesquelles il faut être prudent. Comme nous avons répondu « non », une partie des blocs n'est pas touchée (les **s** comme *skip* sur l'avant-dernière ligne).

Manipuler un fichier binaire pour y mettre des données NDEF ne fait pas partie de nos motivations. Malheureusement, il n'y a pas d'équivalent à **mifare-classic-read-ndef** et **mifare-classic-write-ndef** pour les Mifare Ultralight (mais pour les DESFire oui).

Pour l'écriture NDEF, il n'y a pas de solution parmi les outils de la LibNFC ou de la libfreefare, il faudrait écrire un nouvel utilitaire. Pour la lecture en revanche, un **mifare-ultralight-read-ndef** existe, mais il ne se trouve pas parmi les exemples. Le code est écrit par Romain Tartiere et repris sur le dépôt GitHub d'Eric Betts. Nous l'avons repris et extrait des sources de la libfreefare pour pouvoir le compiler séparément.



Nous avons ici à gauche une carte Mifare Classic 4k, au centre une DESFire EV1 8k et à droite une Mifare Ultralight. Voyez-vous le problème ? Les outils disponibles sur Raspberry Pi prennent en charge les trois tags et permettent également de les différencier (mais pas avec autant de précisions que certaines applications Android).

Celui-ci a été placé dans le dépôt GitHub du magazine (**https://github.com/Hackable-magazine**) avec des autres fichiers accompagnant chaque numéro. Il vous suffira donc de récupérer deux fichiers : **mifare-ultralight-read-ndef.c** et **Makefile**. Une fois placés dans un répertoire, et si vous avez suivi les étapes concernant la prise en charge NFC/RFID sur Raspberry Pi, un simple **make** compilera le tout et vous aurez l'outil **mifare-ultralight-read-ndef** à votre disposition.

Contrairement à **mifare-classic-read-ndef**, celui-ci n'a pas besoin de **ndef-decode** et fait tout le travail tout seul. Nous avons sensiblement modifié l'outil de façon à limiter les messages affichés (regarder les sources, avec un peu d'effort le « vrai » C n'est pas plus compliqué qu'un croquis Arduino). Un tag dont le contenu sera produit par exemple par l'application Android NXP TagWriter ou NFC Tools et ayant un enregistrement de type URI pour alors être lu facilement :

```
$ ./mifare-ultralight-read-ndef
URI=file://musique.mp3
```

L'utilitaire est exécuté directement dans le répertoire courant (**./**) et n'a pas besoin d'être installé. La sortie, comme dans le cas des Mifare Classic pourrait être traitée par **grep** et **sed** de façon plus simple (il y a moins de choses à éliminer et tout apparaît sur stdout).

4. ALLER PLUS LOIN

Nous nous en sommes ici tenus au strict minimum et aux outils les plus simples à utiliser mais, vous vous en doutez, il y a bien des choses à dire et à faire. Nous n'avons, par exemple, pas parlé des Mifare DESfire qui peuvent être utilisés comme tags NFC de type 4, mais sont bien plus que de simples mémoires. L'utilité initiale des exemples des bibliothèques est de vous apprendre à développer des programmes. C'est 90% de l'intérêt de ces outils et nous n'avons donc, en les utilisant simplement, profité que des quelques 10% restants.

Voici quelques autres points intéressants concernant les outils disponibles et les technologies utilisées :

- Certains tags compatibles Mifare Classic, Ultralight et DESFire possèdent une fonctionnalité particulière : la possibilité d'écrire partout dans

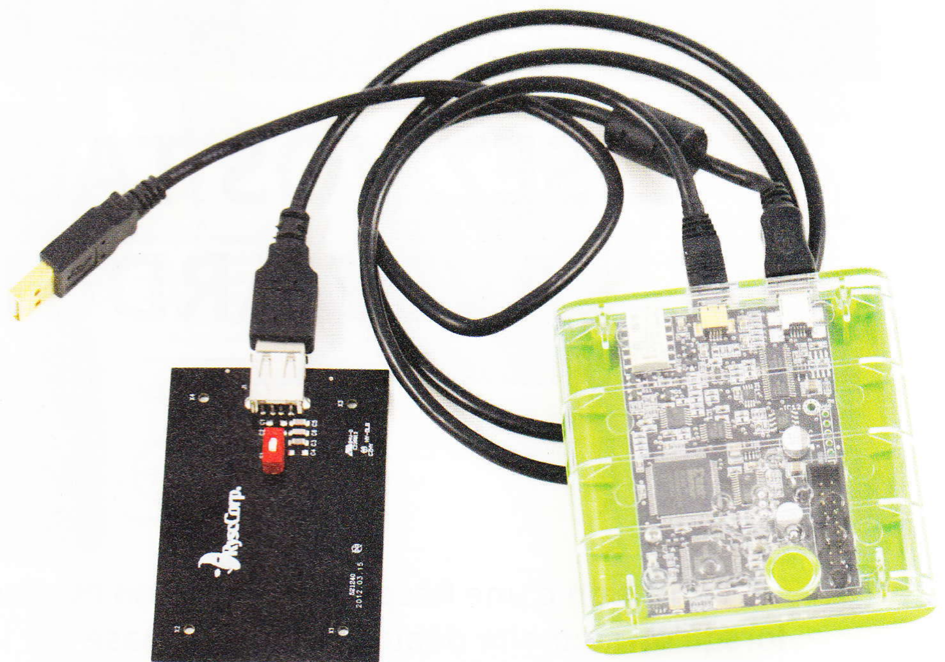
la mémoire et donc de changer l'UID. Deux types existent : ceux permettant tout simplement l'écriture et ceux demandant des manipulations particulières. Ce second cas est couvert par l'outil **nfc-mfsetuid**.

- La sécurité des tags Mifare Classic ou du moins de certains d'entre eux n'est pas absolue. Les outils **mfoc** et **mfcuk** permettent d'exploiter des failles de sécurité afin de retrouver les clés secrètes de vos tags en cas de fausse manipulation ou d'oubli. Ces outils ne sont pas installés avec la LibNFC mais les sources sont disponibles au même endroit (GitHub).
- La LibNFC permet bien plus que la simple lecture/écriture sur tags. C'est une base pour étudier l'ensemble des technologies NFC et donc l'émulation de tags et le dialogue pair-à-pair (LLCP). Sont ainsi à votre disposition des outils comme **nfc-emulate-tag**, **nfc-relay-picc** ou encore **nfc-emulate-forum-tag4**. Il existe également la libllcp, autre projet des créateurs de la LibNFC, spécialisée pour LLCP.

Si vous vous intéressez au langage Python, il existe également *nfcpy*, un module qui n'est pas basé sur la LibNFC, mais permettant de manipuler les tags NFC assez facilement. Là, il n'y a pas d'outils clé en main et vous devrez écrire vos programmes, mais *nfcpy* est très complet et prend en charge 16 lecteurs dont le SCL3711 et l'ACR122U (le support matériel est distinct de la LibNFC). Peut-être reviendrons-nous sur le sujet, mais pour l'heure le présent dossier est déjà très technique et fort volumineux.

Enfin, si vous comptez vraiment intensivement vous lancer dans la découverte et l'exploration du domaine RFID et NFC, il existe un périphérique spécialement conçu pour cela : le Proxmark3.

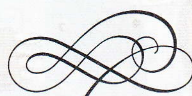
C'est avant tout un outil de développement et d'analyse, disposant de fonctionnalités impressionnantes concernant NFC et RFID (y compris 125/134 Khz) et d'une communauté très active et sympathique. Par contre, ce petit bijou vous coûtera entre $190 et $420 selon la source (chinoise ou officielle) et demande des connaissances techniques non négligeables. Mieux vaut donc y réfléchir à deux fois et à juger posément du budget, du temps et de l'énergie que vous compter investir. C'est un choix personnel et je ne suis sans doute absolument pas objectif, mais je trouve que le jeu en vaut la chandelle, ce sont des jours d'agréables tortures mentales qui vous attendent... **DB**



Le Proxmark3 est un outil générique permettant de manipuler les tags NFC et RFID (basse et haute fréquence). Il est capable de lire et écrire des tags, mais également d'émuler un tag ou d'écouter les transactions entre un lecteur et un tag. C'est un périphérique très puissant, mais également très coûteux, à n'acquérir que si vous comptez réellement vous investir pleinement dans le domaine.

LISEZ VOS TAGS NFC AVEC ARDUINO

Denis Bodor



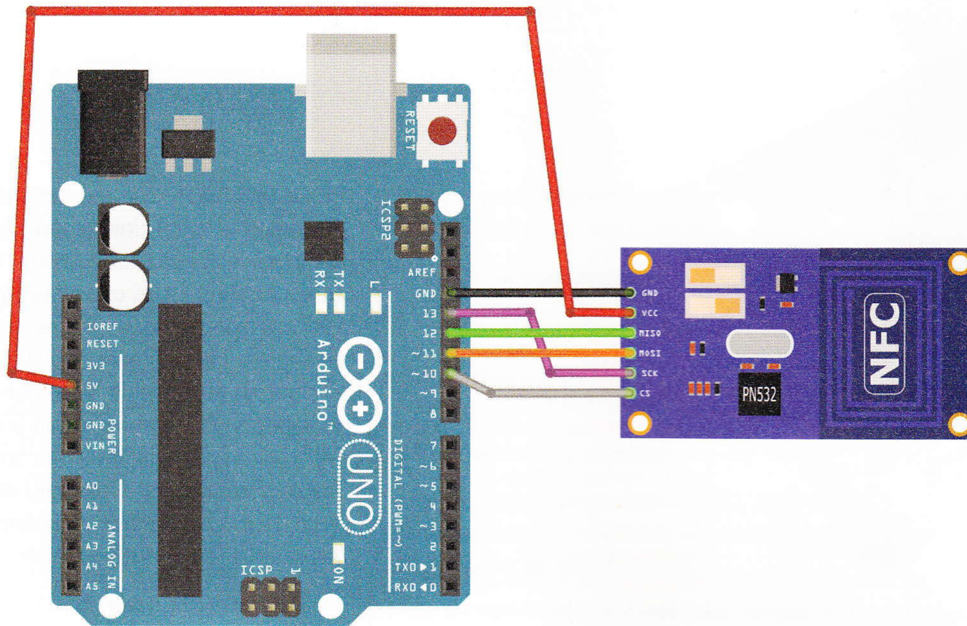
L'utilisation d'une Raspberry Pi ou d'un PC n'est pas une obligation lorsqu'on souhaite débuter un projet basé sur la lecture d'un tag NFC. Une carte Arduino est largement suffisante et sans doute bien plus pratique lorsqu'il s'agit de mettre en place un contrôle d'accès, une automatisation ou tout simplement déclencher une action à la présentation d'un tag. Tout ce qu'il vous faut est un module de lecture.

Il existe deux principaux types de modules lorsqu'il est question de RFID et de NFC avec Arduino. Le premier, très courant se base sur le circuit intégré RC522. Nous n'en parlerons pas ici pour une raison très simple : il ne s'agit pas de NFC, mais de RFID 13,56 MHz. Ils sont parfaitement capables de lire une grande majorité de tags NFC et d'y écrire, mais ne sont pas des périphériques NFC au sens strict du terme, car ne respectant pas intégralement les normes.

L'autre type de module se base sur le PN532 qui lui est effectivement un PCD (*Proximity Coupling Device*) NFC. En termes de lecture/écriture, il offre les mêmes services que le RC522, mais ajoute également les autres fonctionnalités NFC. Matériellement, ce n'est pas d'une grande importance pour un usage de lecture de tags NFC, mais en termes de logiciel cela change tout. En effet, les bibliothèques disponibles pour supporter le RC522 et le PN532, sont très différentes.

La première solution offre des fonctions permettant de lire et écrire la mémoire d'un tag, Mifare Classic en particulier, et la seconde est orientée NFC avec le format de donnée correspondant : le NDEF. Or un des intérêts du NFC est justement de palier au problème de diversité des tags RFID en fournissant un socle unique en termes de communication, mais aussi, et surtout de format de données. Le résultat est clair : le support NDEF ne fonctionne qu'avec le support PN532 (sauf si bien sûr quelqu'un développe un support pour RC522, cela reste techniquement possible).

Choisir NFC, et donc implicitement les modules à base de PN532, vous permettra d'utiliser vos tags indifféremment et simplement avec des applications Android, des outils sur PC et pour vos projets Arduino. Mieux encore, il ne vous sera pas nécessaire de vous pencher sur la technologie d'un tag en particulier et sur la façon dont celui-ci est structuré pour accueillir des données NDEF.



Le duo Arduino et module NFC/PN532 est très simple à assembler sans avoir à dépenser plus que nécessaire en optant pour un shield. Nous avons ici une connexion SPI, mais ce sera tout aussi aisé en i2c.

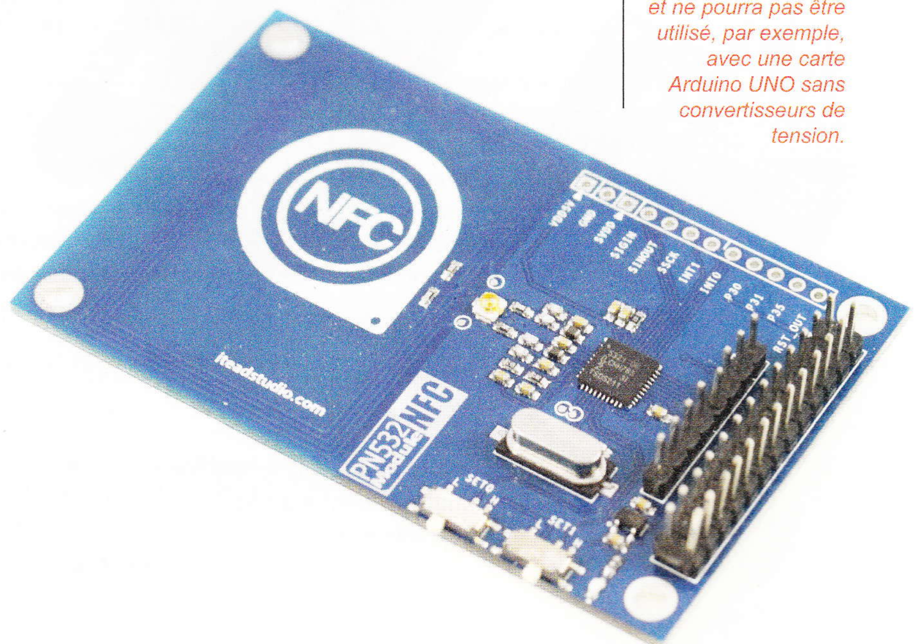
Les tags Mifare Classic utilisent des secteurs et des blocs pour stocker les données, mais les Mifare Ultralight ou encore les NTAG utilisent des pages de 4 octets. Ceci sera totalement transparent pour vous puisque vous raisonnerez en termes de messages et d'enregistrements NDEF. Le format même des données sera également entièrement structuré comme nous l'avons vu dans l'introduction du dossier et cela assurera une compatibilité entre les applications PC/Mac, Android et vos croquis Arduino.

1. LE MODULE

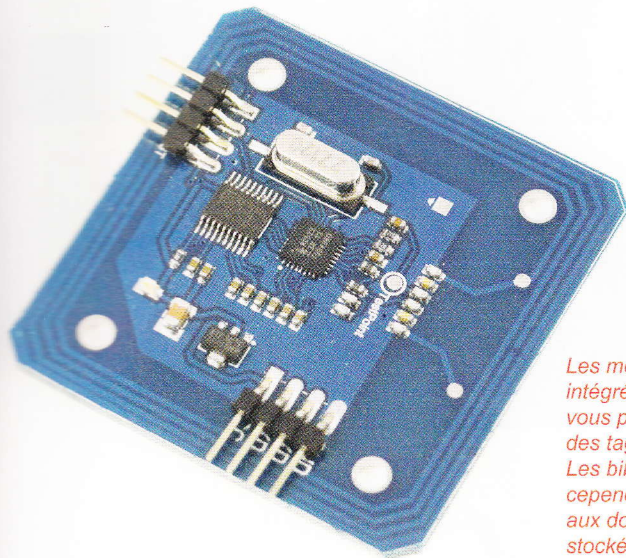
Un module à base de NXP PN532 est relativement facile à trouver. La plupart des modèles affichent clairement l'utilisation du composant principal, ce qui n'est généralement pas le cas des modules à base de RC522. Dans le doute, si un produit ou un vendeur ne précise pas l'utilisation du PN532, mieux vaut éviter l'achat, même s'il est fait mention de NFC et non de RFID.

Le laxisme dans l'utilisation de l'un et l'autre terme, que nous avons vu dans l'introduction, existe également chez les vendeurs.

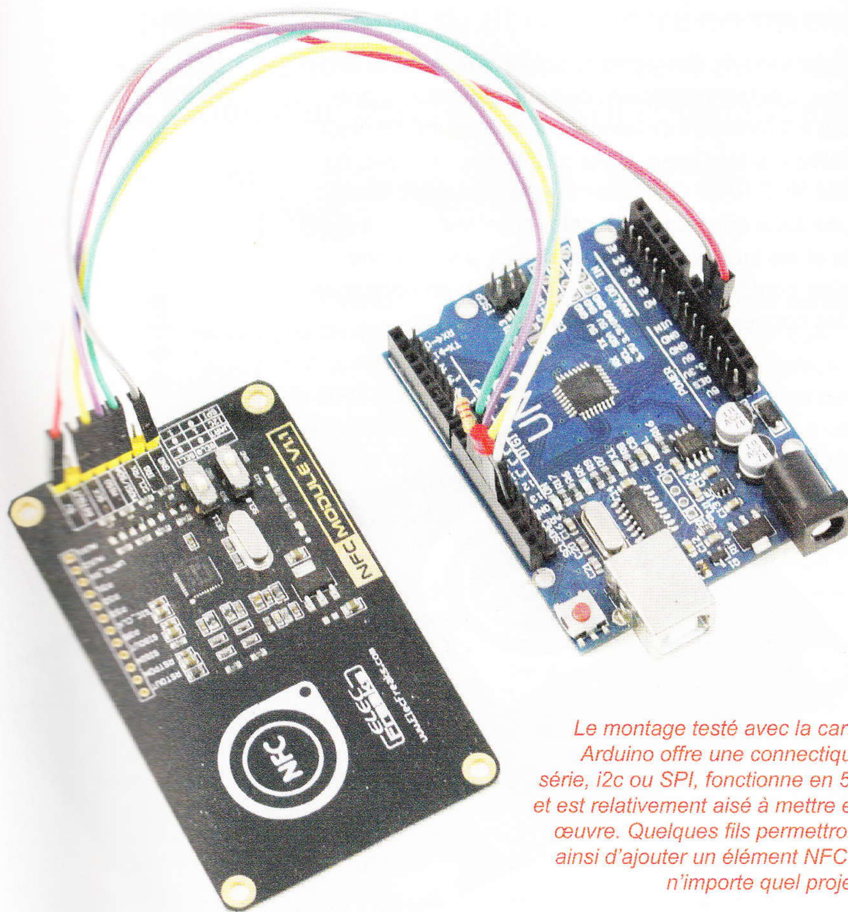
Un tel module devrait vous coûter une quinzaine d'euros. Il existe également des shields Arduino proposant les mêmes fonctionnalités avec le même circuit intégré à leur base. C'est le cas, par exemple, du *PN532 NFC/RFID Controller Shield* d'Adafruit. Bien plus coûteux (plus de 30€), celui-ci présente les avantages et les inconvénients d'un shield, à savoir une certaine compacité (pas de câble), mais, en contrepartie, des connexions et une interface fixe.



Voici un exemple de module à base de PN532 et méritant donc sa mention « NFC ». Celui-ci est l'un des deux modules testés, et provient d'iTeadStudio. Il n'est en revanche pas compatible 5V et ne pourra pas être utilisé, par exemple, avec une carte Arduino UNO sans convertisseurs de tension.



Les modules à base de circuits intégrés RC522, comme celui-ci, vous permettront de lire et écrire des tags compatibles NFC. Les bibliothèques disponibles cependant se limitent à l'accès aux données « brutes » stockées sur les tags. Le décodage des données NDEF sera alors à votre charge.



Le montage testé avec la carte Arduino offre une connectique série, i2c ou SPI, fonctionne en 5V et est relativement aisé à mettre en œuvre. Quelques fils permettront ainsi d'ajouter un élément NFC à n'importe quel projet.

Le PN532 permet en effet une connexion via trois interfaces : SPI, i2c et série. La plupart des modules permettent de choisir le type de connexion via des cavaliers ou des micro-interrupteurs (*dip switch*) et donc d'intégrer plus facilement le support NFC à un nouveau projet ou un projet existant. Avec un shield comme celui d'Adafruit, ce sera i2c, point (à moins de bidouiller, mais on perd l'intérêt d'un shield).

Un autre point très important à propos de ces modules concerne les tensions utilisées. Certains d'entre eux sont interfacés en 3,3V et d'autres en 5V. Les premiers seront adaptés pour une utilisation avec Raspberry Pi, Arduino Zero, Arduino Due, Ti Launchpad, etc., et les seconds pour le reste de la gamme Arduino comme la très populaire UNO par exemple. Tout comme pour la différenciation RC522/PN532, si la description du produit ne précise pas clairement la tension utilisée, évitez simplement l'achat.

2. CONNEXION

Nous utiliserons le module en SPI. Sur le modèle utilisé, ceci correspond à une position des interrupteurs en SEL0 sur GND et SEL1 sur 3,3V (notez que la valeur sérigraphiée se réfère à la tension interne au module). À toutes fins utiles, le module de test en question est fabriqué par ElecFreaks et est dénommé « NFC Module V1.1 ». Un autre modèle en ma possession est celui de iTead Studio, en 3,3V, et là encore la position des interrupteurs pour

une utilisation en SPI est SEL0 sur « L » (*low*, la masse) et SEL1 sur « H » (*high*, Vcc).

Ces interrupteurs correspondent aux broches 16 (SEL0) et 17 (SEL1) du PN532. Les combinaisons possibles sont :

- 0 et 0 : liaison série (RX/TX) ;
- 1 et 0 : i2c (SDA, SCL) ;
- 0 et 1 : SPI (MISO, MISO, SCK, /CS).

Du fait du choix d'interface à configurer, certaines broches ont plusieurs usages. Il est important de le noter, car tantôt les fabricants de modules ne marquent qu'une seule nomenclature concernant les broches. Ainsi, nous avons les broches suivantes utilisées sur un PN532 :

- 27 : CS (SPI) / RX (série) / SCL (i2c) ;
- 28 : MOSI (SPI) / TX (série) / SDA (i2c) ;
- 29 : MISO (SPI) ;
- 30 : SCK (SPI).

Certains modules fournissent également la broche IRQ permettant entre autres choses de réveiller le PN532 s'il est en sommeil ou dans le sens inverse de réveiller le montage si un événement NFC survient. Ceci est particulièrement utile dans un contexte de réduction d'énergie, comme le fonctionnement sur accu ou pile afin de maximiser l'autonomie. Ceci ne sera pas traité dans le présent article.

Enfin, concernant la connexion i2c, si tel est votre choix, n'oubliez pas d'ajouter les résistances de rappel entre 2,2 Kohm et 10 Kohm entre les lignes SDA/SCL et la tension d'alimentation, ainsi qu'éventuellement un condensateur de filtrage entre l'alimentation et la masse du module si vos câbles sont longs.

En SPI, la connexion à une carte Arduino UNO est relativement simple :

- MOSI sur 11 ;
- MISO sur 12 ;
- SCK sur 13 ;
- CS (alias /CS, SS ou NSS) sur 10.

En i2c, les broches SDA et SCL sont directement référencées sur les cartes Arduino récentes et correspondent respectivement aux entrées analogiques A4 et A5.

En termes logiciels, mais nous reviendrons plus pleinement sur le sujet, la différence se jouera sur l'initialisation en début d'article. Pour SPI, nous avons :

```
#include <SPI.h>
#include <PN532_SPI.h>
#include <PN532.h>
#include <NfcAdapter.h>

PN532_SPI pn532spi(SPI, 10);
NfcAdapter nfc = NfcAdapter(pn532spi);
```

Et pour l'i2c :

```
#include <Wire.h>
#include <PN532_I2C.h>
#include <PN532.h>
#include <NfcAdapter.h>

PN532_I2C pn532_i2c(Wire);
NfcAdapter nfc = NfcAdapter(pn532_i2c);
```

Après ces quelques lignes, `nfc` est le seul objet que nous manipulons pour dialoguer avec le module PN532 et l'aspect « interface » ne rentre plus en ligne de compte. Il sera donc très facile de basculer un projet de SPI en i2c et inversement, ce qui est plus utile, je pense, lorsqu'on commence à ajouter des éléments comme un écran LCD, une RTC, un afficheur Oled, etc. (le nombre de broches disponibles est inversement proportionnel à l'imagination du programmeur).

3. BIBLIOTHÈQUE ET CROQUIS DE DÉPART

La bibliothèque à utiliser pour prendre en charge le module à base de PN532 n'est pas disponible (pour l'instant) via le gestionnaire de l'IDE Arduino. Il vous faudra vous rendre sur la page **https://github.com/don/NDEF**.

Là, vous y trouverez le résultat du travail de *Don Coleman* (également co-auteur d'un ouvrage sur NFC chez O'Reilly) et en particulier sa bibliothèque NDEF qui, contrairement à ce que son nom laisse penser, ne se limite pas au traitement des données NDEF, mais à l'ensemble du support NFC à l'exception de la prise en charge directe du matériel. Cette partie est l'affaire d'une autre bibliothèque, écrite par *Yihui Xiong*, mais diffusée à présent via le GitHub de Seeed Studio (cette société vend également shields et modules NFC compatibles) : **https://github.com/Seeed-Studio/PN532**.

Attention, il existe une bibliothèque PN532 dans le gestionnaire de bibliothèques. Celle-ci est l'œuvre d'Adafruit pour le support de son propre matériel. Je n'ai pas expérimenté outre mesure dans cette direction, m'intéressant principalement aux fonctionnalités de la bibliothèque de *Don Coleman* et au support NDEF. La bibliothèque Adafruit ne fonctionne pas de la même manière que celle de Seeed Studio et ne répond pas à la même logique interne. Tâchez simplement de ne pas mélanger les deux supports PN532.

L'installation des bibliothèques est relativement simple, même si elle doit être faite manuellement et qu'il existe quelques subtilités. Commençons par le support PN532. Depuis la page GitHub, récupérez le fichier **PN532-master.zip** en cliquant sur le bouton ***Download ZIP*** en haut à droite. Cette archive contient un répertoire **PN532-master**, contenant lui-même les éléments qui nous intéressent. Copiez alors les répertoires **PN532**, **PN532_HSU**, **PN532_I2C** et **PN532_SPI**, et leur contenu, dans le sous-répertoire **libraries** de votre dossier correspondant au carnet de croquis (spécifié dans les préférences de l'environnement Arduino).

Allez ensuite sur la page GitHub de la bibliothèque de *Don Coleman* et récupérez de la même façon le fichier **NDEF-master.zip**. Celui-ci contient également un répertoire, **NDEF-master**, possédant les fichiers utiles. Il vous faudra en revanche copier ce répertoire et son contenu dans votre **libraries** tout en le renommant **Ndef**.

Une fois toutes ces opérations effectuées, le résultat doit ressembler à ceci :

```
libraries/
   PN532/
   PN532_HSU/
   PN532_I2C/
   PN532_SPI/
   Ndef/
```

Dès lors, vous serez prêt à lancer votre l'IDE Arduino pour écrire votre premier croquis :

```
// support SPI
#include <SPI.h>
// support PN532 sur SPI
#include <PN532_SPI.h>
// support générique PN532
#include <PN532.h>
// support NFC
#include <NfcAdapter.h>

// lecteur en SPI avec /CS sur 10
PN532_SPI pn532spi(SPI, 10);
// Le lecteur est ce nouvel objet
NfcAdapter nfc = NfcAdapter(pn532spi);

void setup(void) {
  Serial.begin(115200);
  Serial.println("Lecteur NDEF");
```

```
  // Initialisation et affichage du lecteur détecté
  nfc.begin();
}

void loop(void) {
  Serial.println("\nApprochez un tag NFC");
  // Si ne détecte aucun tag on s'arrête là
  while(!nfc.tagPresent()) {
    delay(5);
    return;
  }
  // Un tag est trouvé
  Serial.println("Je vois un tag ");
    // Lecture
  NfcTag tag = nfc.read();
  // test si enregistrement NDEF présent
  if (tag.hasNdefMessage()) {
    Serial.println("contenant du NDEF");
    // récupération du contenu NDEF
    NdefMessage message = tag.getNdefMessage();
    // S'il y a au moins un message
    if (message.getRecordCount() > 0) {
      Serial.println("avec au moins un message NDEF");
      // On récupère le premier enregistrement
      NdefRecord record = message.getRecord(0);
      // Nous ne voulons que des types bien connus
      if(record.getTnf()==TNF_WELL_KNOWN) {
        Serial.println("ayant le TNF \"bien-connu\"");
        // et que des URIs
        if(record.getType()=="U") { // URL
          Serial.println("type URI");
          // récupération de la taille du contenu
          int payloadLength = record.getPayloadLength();
          // variable pour le contenu
          byte payload[payloadLength];
          // récupération du contenu
          record.getPayload(payload);
          // Fabrication d'un objet String de caractères
          String mesdonnees = "";
          // Le vrai contenu commence à la position 1, non 0
          // À 0, il y a le code identifiant de l'URI
          for (int i = 1; i < payloadLength; i++) {
            mesdonnees += (char)payload[i];
          }
          // Le premier correspond à une URI file:// ?
          if(payload[0]==0x1D ) {
            Serial.print("file://");
            Serial.println(mesdonnees);
          // si non, on s'intéresse aussi aux URI personnalisées
          } else if(payload[0]==0x00) {
            Serial.println(mesdonnees);
          }}}}}
  delay(1000);
}
```

L'ensemble est relativement linéaire et nous procédons par étape. Nous avons déjà parlé de l'initialisation plus haut dans l'article. Après cette étape, tout passera par l'objet `nfc` de type `NfcAdapter` que nous initialisons avec la méthode `begin()` comme c'est le cas avec de nombreuses autres bibliothèques (comme « LiquidCrystal » par exemple). Notez que par défaut cette initialisation provoquera l'affichage d'informations sur le moniteur série :

```
Found chip PN532
Firmware ver. 1.6
```

Dans notre cas, ce n'est pas un problème puisque nous comptons justement utiliser cette fonctionnalité pour fournir d'autres informations. Si votre projet n'utilise pas `Serial`, vous pouvez utiliser `nfc.begin(false)`. Le paramètre passé en argument désactive l'affichage « verbeux » (à `true` par défaut).

Le reste se passe intégralement dans la boucle `loop()` avec en premier lieu une autre boucle testant la valeur retournée par `tagPresent()` qui est VRAI si le lecteur a détecté un tag et FAUX dans le cas contraire. Ainsi tant que le résultat est FAUX (différent de zéro), notre `while` se contentera de mettre fin, après une courte pause, à la fonction `loop()` qui repart pour un tour.

Si `tagPresent()` retourne VRAI en revanche, nous poursuivons en déclenchant la lecture du tag avec `read()` afin d'obtenir un objet `NfcTag` représentant le tag que nous appelons simplement `tag`. Le tag et ses informations sont dès lors accessibles via cette variable.

Nous pouvons alors utiliser la méthode `hasNdefMessage()` pour nous enquérir de la présence de données NFC/NDEF sur le tag. Là encore, si la méthode retourne VRAI c'est que le tag répond à nos attentes et nous récupérons le message en question sous la forme d'un objet `message` de type `NdefMessage`. Rappelons ici qu'un message NDEF est une unité de communication et qu'il peut contenir plusieurs enregistrements NDEF.

Nous utilisons alors la méthode `getRecordCount()` sur l'objet `message` pour connaître le nombre d'enregistrements NDEF qui composent le message. Ici, nous partons du principe que seul le premier enregistrement est important et vérifions donc simplement si le nombre est un ou plus.

On imaginera aisément les applications possibles dans le cas d'une succession de plusieurs enregistrements, pour stocker, par exemple, des états successifs sur une sortie de l'Arduino ou encore différentes valeurs pour différentes sorties analogiques/PWM.

Pour récupérer un enregistrement, il suffit d'utiliser la méthode `getRecord()` avec, en argument, le numéro de l'enregistrement (en commençant par `0` bien sûr). Ici, seul le premier nous intéresse, mais pour récupérer tous les enregistrements, une boucle `for` ferait parfaitement l'affaire. L'enregistrement est stocké dans `record`, un objet de type `NdefRecord` dont, là encore, nous pouvons utiliser certaines méthodes. `getTnf()` par exemple nous informera sur le TNF (type de type) de l'enregistrement. Nous ne voulons que le TNF « bien connu » (*Well-Known*, cf. article d'introduction) et un enregistrement de type `U` comme URI, que nous vérifions avec `getType()`.

À partir de là, il ne nous reste plus qu'à récupérer les véritables données enregistrées, autrement dit, la charge utile (*payload* en anglais). Comme nous ne nous intéressons qu'au type URI, nous savons que le premier octet de ces données définit le code identifiant de l'URI. Ici, nous nous intéressons à deux possibilités : `0x1d` correspondant à `file://` ou `0x00` signifiant l'absence de code.

Cette dernière option nous permet de l'intégrer manuellement dans les données. C'est fort pratique pour des URI qui ne sont pas dans la liste des codes prédéfinis (`http://`, `tel:`, `mailto:`, etc.). Nous pouvons, par exemple, décider d'utiliser `port://` suivi d'un numéro de port PWM et d'une valeur. Ainsi, `port://5/64` pourrait signifier « port 5 PWM à 25% » (64/255). Bien entendu, ceci nous fait consommer 7 octets de plus par enregistrement (puisque nous utilisons de toute façon le code `0x00` en guise de premier octet de l'enregistrement).

Nous pouvons également « tricher » en utilisant, comme ici, un code URI qui n'a aucun rapport avec le type de données. `file://5/64` peut être également interprété par notre croquis comme « port 5 PWM à 25% », ce qui ne fait que 5 octets en tout. Mais nous sommes alors hors standard, car `file://` est fait pour désigner un chemin et un fichier, concept inexistant dans un croquis Arduino (sauf si on accède à une carte SD, mais c'est là un sujet pour un autre article).

Le résultat de l'exécution du croquis nous donne, sur le moniteur série, quelque chose comme ceci :

```
Approchez un tag NFC
Je vois un tag
contenant du NDEF
avec au moins un message NDEF
ayant le TNF "bien-connu"
type URI
file://13/1
```

Le tag en question a été effacé et écrit par l'application NFC Tool sous Android, mais le même résultat peut être obtenu avec n'importe quelle autre solution Android, PC ou Raspberry Pi.

Le tag de cet essai est un autocollant Mifare Ultralight, mais le résultat est exactement le même avec ces mêmes données NDEV placées sur un NTAG213 ou encore un Mifare Classic 1K. Tout l'avantage de ces bibliothèques est précisément là : vous n'avez pas à vous soucier de la technologie sous-jacente et du format de stockage effectif des données. Les seules informations utiles dépendent du NDEF et c'est là l'objet même du NFC et de ses standards.

4. FAIRE QUELQUE CHOSE D'UTILE

Nous n'avons pas détaillé la partie la plus importante de notre croquis. Une fois que nous avons accès aux données utiles, nous récupérons leur taille en octet avec `getPayloadLength()` et utilisons cette valeur pour créer un tableau de `byte` de la bonne taille. Celui-ci est utilisé ensuite en argument de `getPayload()` et se retrouve initialisé avec une copie des données :

```
int payloadLength = record.getPayloadLength();
byte payload[payloadLength];
record.getPayload(payload);
```

Afin de pouvoir manipuler plus aisément ces données, nous construisons alors un objet `String` (chaîne de caractères) en prenant soin de sauter le premier caractère :

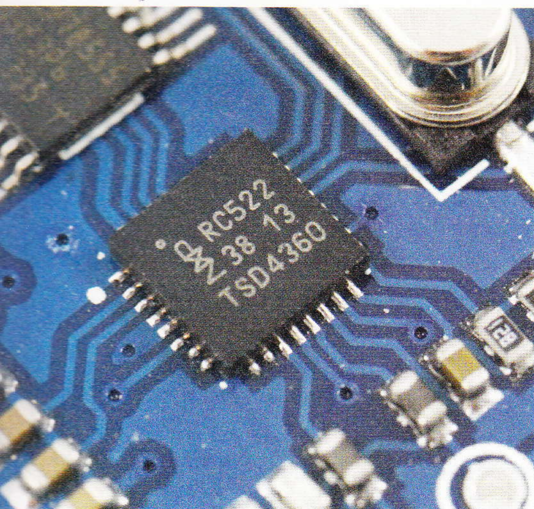
```
String mesdonnees = "";
for (int i = 1; i < payloadLength; i++) {
  mesdonnees += (char)payload[i];
}
```

`mesdonnees` pourra ensuite être utilisé comme n'importe quelle chaîne et, bien entendu, être envoyé au moniteur série comme nous le faisons un peu plus loin.

L'étape suivante consistera alors simplement à traiter cette chaîne de façon à en extraire ce qui nous intéresse. Si nous choisissons d'utiliser une URI comme `file://port/valeur` pour nos tags, nous pouvons très simplement apporter les modifications. Notre condition `if` de départ :

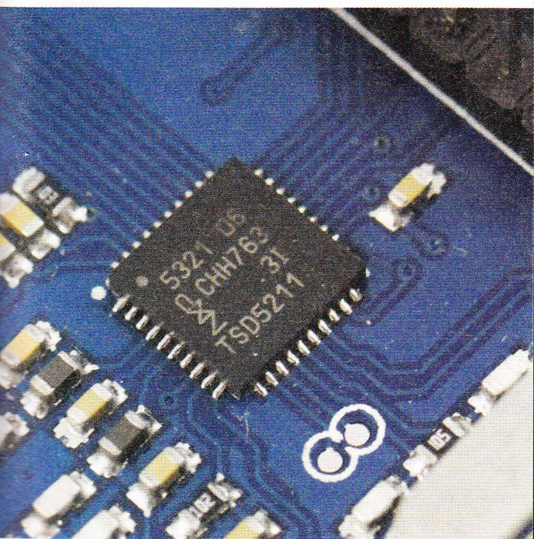
```
if(payload[0]==0x1D ) {
  Serial.print("file://");
  Serial.println(mesdonnees);
}
deviendra :
if(payload[0]==0x1D ) {
  int port = mesdonnees.substring(0,mesdonnees.indexOf('/')).toInt();
  int val = mesdonnees.substring(mesdonnees.indexOf('/')+1).toInt();
  pinMode(port, OUTPUT);
  digitalWrite(port, val);
}
```





Le circuit intégré RC522 de NXP est très populaire, en particulier pour les modules les moins chers. Il conviendra cependant de l'éviter si l'on s'intéresse aux technologies « véritablement » et « totalement » NFC. Mais pour un simple usage de lecture/écriture de tags RFID 13,56 mhz au plus bas niveau, il conviendra très bien.

Gros plan sur LE composant important de tout module NFC, le circuit intégré PN532 de NXP chargé non seulement de la communication avec les tags, mais également avec la carte hôte (Arduino en l'occurrence) qui le prend en charge.



Certes, je reconnais que présenté comme ça c'est un peu violent. Mais il suffit de découper, comme souvent, en petits morceaux.

Nous avons au départ une chaîne de caractères, un objet `String` nommé `mesdonnees`. Il existe tout un tas de méthodes associées à cette classe d'objets, l'une d'elles `indexOf()` nous permet d'obtenir la position du premier caractère spécifié en argument (ici `/`).

`mesdonnees.indexOf('/')` est donc la valeur numérique d'une position dans la chaîne. Mais nous avons également la méthode `substring()` qui permet d'obtenir un morceau de la chaîne de départ (sous-chaîne ou *substring* en anglais). Il suffit de lui passer une valeur pour obtenir une chaîne à partir de la position précisée, ou deux valeurs pour fixer une plage.

Imaginons que notre chaîne soit `"Bonjour"`. `indexOf('j')` sera `3`, `substring(2)` nous donnera `"jour"` et `substring(0,3)` donnera `"Bon"`. Nous pouvons alors combiner les deux méthodes pour obtenir la sous-chaîne souhaitée. Mais comme le résultat est une chaîne et que dans notre cas nous avons besoin d'une valeur numérique entière (`int`), nous utilisons, en plus, une autre méthode : `toInt()`. Celle-ci transforme le texte en valeur ou, autrement dit, un chiffre en nombre.

On combine alors joyeusement tout cela pour extraire, par exemple `7` et `1` de `7/1` (n'oubliez pas, le `file://` n'est pas réellement présent, c'est juste un code en début des données utiles qui n'est pas dans notre chaîne). Nous récupérons ces valeurs dans `port` et `val` et les utilisons avec `pinMode()` et `digitalWrite()`. Ainsi la lecture d'un tag contenant `file://7/1` passera le port 7 à l'état haut et `file://7/0` le passera à l'état bas. Connectez une led sur le port avec une résistance et vous voici en mesure de l'allumer et l'éteindre en passant des tags.

Bien entendu, il s'agit ici d'une simple démonstration. Pour une utilisation pratique réelle, il faudra tester les valeurs dans `port` et `val` avant de les utiliser. Lorsqu'on lit des données qui proviennent de l'extérieur, il faut TOUJOURS les valider avant usage. En effet, qu'arriverait-il si un tag contenait `file://42/800` ?

5. LE RESTE, C'EST POUR VOUS !

Partant de cet exemple d'un simple port en bascule, il est possible d'extrapoler. Pourquoi ne pas utiliser une sortie PWM ? Quid de l'utilisation d'une led type WS2812b (voir *Hackable n°6*) avec des valeurs RVB sur un tag ? Pourquoi pas la position d'un relais, celle d'un servomoteur ou encore un texte à afficher sur un écran LCD ?

Ce n'est qu'une piste à suivre. Jetez un œil aux exemples livrés avec la bibliothèque NDEF, vous y trouverez de tout. Y compris de quoi utiliser le module PN532 en NFC pair-à-pair. Et là, cela prendra une tout autre dimension ! **DB**

HACKABLE MAGAZINE

DÉCOUVREZ NOS OFFRES D'ABONNEMENTS !

PRO OU PARTICULIER = CONNECTEZ-VOUS SUR :

www.ed-diamond.com

LES COUPLAGES PAR SUPPORT :

VERSION **PAPIER**

Retrouvez votre magazine favori en papier dans votre boîte à lettres !



VERSION **PDF**

Envie de lire votre magazine sur votre tablette ou votre ordinateur ?



SÉLECTIONNEZ VOTRE OFFRE DANS LA GRILLE AU VERSO ET RENVOYEZ CE DOCUMENT COMPLET À L'ADRESSE CI-DESSOUS !

Voici mes coordonnées postales :	
Société :	
Nom :	
Prénom :	
Adresse :	
Code Postal :	
Ville :	
Pays :	
Téléphone :	
E-mail :	

☐ Je souhaite recevoir les offres promotionnelles et newsletters des Éditions Diamond.

☐ Je souhaite recevoir les offres promotionnelles des partenaires des Éditions Diamond.

En envoyant ce bon de commande, je reconnais avoir pris connaissance des conditions générales de vente des Éditions Diamond à l'adresse internet suivante : boutique.ed-diamond.com/content/3-conditions-generales-de-ventes et reconnais que ces conditions de vente me sont opposables.

HACKABLE MAGAZINE

Les Éditions Diamond
Service des Abonnements
10, Place de la Cathédrale
68000 Colmar – France
Tél. : + 33 (0) 3 67 10 00 20
Fax : + 33 (0) 3 67 10 00 21

Vos remarques :

VOICI TOUTES LES OFFRES COUPLÉES AVEC HACKABLE !

POUR LE PARTICULIER ET LE PROFESSIONNEL ...

Prix TTC en Euros / France Métropolitaine

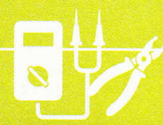
CHOISISSEZ VOTRE OFFRE !

SUPPORT		PAPIER		PAPIER + PDF		PAPIER + BASE DOCUMENTAIRE		PAPIER + PDF + BASE DOCUMENTAIRE	
Prix en Euros / France Métropolitaine				PDF 1 lecteur		1 connexion BD		PDF 1 lecteur + 1 connexion BD	
Offre	ABONNEMENT	Réf	Tarif TTC	Réf	Tarif TTC	Réf	Tarif TTC	Réf	Tarif TTC
HK	6n° HK*	☐ HK1	39,-	☐ HK12	58,-				
	LES COUPLAGES « EMBARQUÉ »								
D	6n° HK* + 4n° OS	☐ D1	65,-	☐ D12	98,-	☐ D13	85,-*	☐ D123	118,-*
E	6n° HK* + 4n° OS + 6n° MISC	☐ E1	105,-	☐ E12	158,-	☐ E13	179,-*	☐ E123	232,-*
E+	6n° HK* + 4n° OS + 6n° MISC + 2n° HS	☐ E+1	119,-	☐ E+12	179,-	☐ E+13	193,-*	☐ E+123	253,-*
F	6n° HK* + 4n° OS + 11n° GLMF	☐ F1	125,-	☐ F12	188,-	☐ F13	229,-*	☐ F123	292,-*
F+	6n° HK* + 4n° OS + 11n° GLMF + 6n° HS	☐ F+1	183,-	☐ F+12	275,-	☐ F+13	287,-*	☐ F+123	379,-*
G	6n° HK* + 4n° OS + 6n° LP	☐ G1	100,-	☐ G12	150,-	☐ G13	164,- *	☐ G123	214,-*
G+	6n° HK* + 4n° OS + 6n° LP + 3n° HS	☐ G+1	129,-	☐ G+12	194,-	☐ G+13	193,-*	☐ G+123	258,-*
	LES COUPLAGES « GÉNÉRAUX »								
H	6n° HK* + 4n° OS + 6n° LP + 6n° MISC + 11n° GLMF	☐ H1	200,-	☐ H12	300,-	☐ H13	402,-*	☐ H123	499,-*
H+	6n° HK* + 4n° OS + 6n° LP + 3n° HS + 6n° MISC + 2n° HS + 11n° GLMF + 6n° HS	☐ H+1	301,-	☐ H+12	452,-	☐ H+13	493,-*	☐ H+123	639,-*

Les abréviations des offres sont les suivantes : LM = GNU/Linux Magazine France | HS = Hors-Série | LP = Linux Pratique | OS = Open Silicium | HC = Hackable

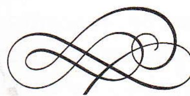
* HK : Attention : La base Documentaire de Hackable n'est pas incluse dans l'offre.

N'hésitez pas à consulter les détails des offres ci-dessus sur : www.ed-diamond.com !

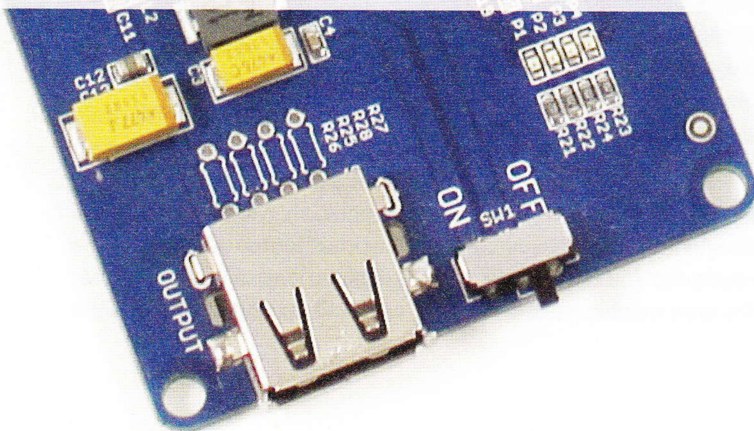


LIPO RIDER PRO : L'AUTONOMIE SOLAIRE CLÉ EN MAIN POUR VOS PROJETS

Denis Bodor



L'énergie gratuite est le rêve de tout le monde et de l'énergie nous en avons à volonté dès lors qu'on ne vit pas dans une caverne ou une grotte : c'est le soleil. Les choses cependant ne sont pas si simples, car si utiliser des cellules photovoltaïques peut sembler aisé à première vue, les choses se compliquent vraiment dès lors qu'on souhaite charger une batterie. Pour vos montages électroniques, il existe une solution clé en main et open hardware : le LiPo Rider Pro.







Voici une batterie d'aéromodélisme pour mon multirotor DJI F550, celle-ci est marquée « 4S1P 14,8V 35C 129.5 Amp ». Elle se compose donc de 4 cellules ou accumulateurs de 3,7V assemblés en série, fournissant 14,8V. Elle est en mesure de fournir un courant continu de 35C, soit 3,7×38 ampères, ou 129,5 A. Parfait pour satisfaire 6 moteurs brushless fonctionnant de concert pour faire voler mon aéronef.

Charger une batterie ou un accumulateur (ou « accu ») est un processus qu'on peut qualifier de délicat, car en fonction du type de matériel, cela consiste principalement à fournir un courant à une certaine tension et à surveiller le comportement de l'ensemble de façon à déterminer les différentes étapes de charge. Certains accumulateurs sont plus tolérants que d'autres, mais la technologie privilégiée actuellement, les LiPo (*Lithium Polymère*), ne pardonne pas. Ces accus sont légers, compacts et économiques, mais ils sont également fragiles et très dangereux dès lors qu'on ne les respecte pas suffisamment. Ici, le terme « dangereux » doit clairement être vu comme synonyme de « dégagement gazeux », « feu », « incendie », « explosion », etc. Les amateurs d'aéromodélisme le savent bien et chargent/stockent leurs batteries dans des pochettes de sécurité, voire dans des parpaings creux (agglo) et noyés dans du sable (cherchez simplement « lipo fire » sur YouTube, vous verrez, on redevient vite un adulte responsable).

D'autre part, nous avons le soleil, une boule de 1,4 million de kilomètres de diamètre, se trouvant à quelques 150 millions de kilomètres de nous. Ce gros machin génère une énergie phénoménale (3,826 x 10^26 watts), mais, en ce qui nous concerne nous terriens, de façon absolument erratique. En prenant en compte le fait que notre atmosphère est très capricieuse en terme de perméabilité lumineuse, on devine rapidement que nous avons là quelque chose qu'il est relativement difficile d'utiliser pour la charge d'un accu qui demande un courant parfaitement contrôlé. C'est un peu comme essayer de verser l'huile pour faire une mayonnaise sur un trampoline peuplé de kangourous dopés au Red Bull.

Avant de poursuivre, précisons une chose importante. D'un point de vue terminologique :

- Une pile est un élément non rechargeable (oui, je sais, il existe des moyens plus ou moins hasardeux de recharger des « piles », mais soyons clairs : si ça fonctionne, il ne s'agit plus, par la force des choses, d'une pile, mais d'un accumulateur).
- Un accumulateur est un élément capable d'accumuler et restituer de l'énergie via un processus électrochimique (comme les piles) ou électrostatique. Dans ce second cas, on parle de condensateurs, ou de super-condensateurs si leur capacité est comparable aux solutions électrochimiques. Acide/plomb, NiCd (nickel-cadmium), NiMH (nickel-hydrure métallique), li-Ion (lithium-ion), LiFe/LiFePO (lithium fer phosphate),



Ceci est un chargeur SKYRC, capable de charger et décharger plusieurs types de batteries, composé de différents agencements d'accumulateurs. La configuration est réglable via quelques boutons afin d'adapter la charge/décharge en fonction du nombre de cellules et du courant à utiliser.

LiPo... sont autant de technologies utilisées pour les accumulateurs électrochimiques, ayant chacune des avantages et des inconvénients.

- Une batterie est un ensemble d'éléments et plus exactement c'est « une batterie d'accumulateurs ». On parle également de cellules pour décrire les accumulateurs d'une batterie (en particulier à propos des batteries LiPo pour l'aéromodélisme où « 2S », « 3S », « 6S », etc., désigne le nombre de cellules en série).

Notez que par confusion ou parce que la différenciation n'existe pas en anglais, les termes « accumulateurs » (ou « accus ») et « batteries » sont souvent utilisés indifféremment. C'est une erreur. Dans le sens strict du terme, votre smartphone a certainement un accu, mais votre ordinateur portable une batterie, tout comme votre voiture (une batterie 12V au plomb groupant généralement 6 accumulateurs en série fournissant chacun 2V).

1. CONTRÔLE DE CHARGE

Le problème ici est double. Nous avons d'une part l'énergie solaire qui n'est pas constante et de l'autre une charge d'accu qui doit l'être. Mais même en éliminant le problème solaire et en utilisant une source stable, il n'est pas question de simplement la brancher sur l'accu, en particulier avec des LiPo (sauf si vous aimez les explosions, les flammes, les pompiers et les hôpitaux).

Tous les accus ne se chargent pas de la même manière. Le cas qui nous intéresse ici est celui des LiPo et la charge est différente, par exemple, pour une batterie de voiture. Un accumulateur LiPo a une tension nominale de 3,7 volts et une capacité dépendante de sa fabrication et de sa taille. Une batterie LiPo sera constituée de plusieurs accumulateurs ou cellules connectées en série ou en parallèle. Les fabricants indiquent généralement ces informations directement sur la batterie elle-même. Si nous prenons l'exemple d'une batterie de modélisme KyPOM KT3300/45-3S, nous voyons dessus :

- « 3S1P » : nous avons 3 cellules en série (S) et une en parallèle (P), il s'agit donc de trois accus connectés en série (voyez cela comme une grille 3×1).

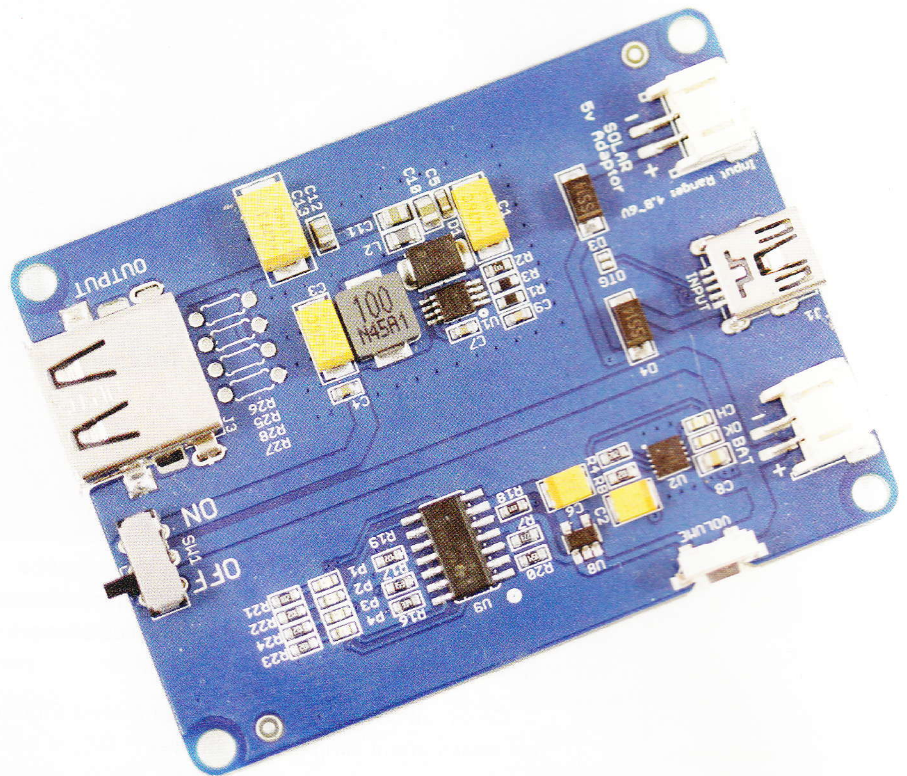
Le stockage et le transport des accumulateurs et batteries LiPo demande de la prudence. Comme d'autres, j'ai pour habitude de placer systématiquement ces composants dans un sac ignifugé ou ne jamais les laisser à proximité de matériaux inflammables. Je trouve les pompiers fort sympathiques, mais je préfère les voir chez moi uniquement lorsqu'ils viennent me vendre leur calendrier (que j'achète systématiquement d'ailleurs).

- « 11.1V » : c'est la tension nominale de la batterie, ce qui correspond effectivement à 3 fois 3,7 volts.
- « 3300 mAh » : ou 3300 milliampères/heure, soit une capacité théorique permettant de fournir 3,3 ampères pendant une heure, ou 330 milliampères pendant 10 heures, 660 milliampères pendant 5 heures, etc. (j'ai bien dit « théorique »).
- « 25C » : là, c'est un peu spécial. Cette mention fait référence au courant qu'il est possible de tirer de façon continue de la batterie. Le « C » correspond à une équivalence en ampères calquée sur la capacité de la batterie. Ici, elle est de 3300 mAh et un C sera donc 3,3 ampères. Comprenez bien qu'il ne s'agit pas d'un calcul, car on ne peut convertir des pommes (Ah) en oranges (A), mais d'une donnée constructeur. Cette batterie permet donc de fournir 25 fois ce courant, soit 82,5 A. Si vous vous pliez d'un petit calcul, à 11,1 volts, cela nous donne plus de 900 watts.



Chose qui n'est pas spécifiée sur la batterie elle-même, le courant de charge pour ce modèle précis est spécifié entre 1C et 3C. Ceci signifie que la batterie peut être chargée avec un courant compris entre 3,3 ampères et 9,9 ampères. Bien entendu, plus le courant de charge est important, moins la charge dure longtemps, mais également, moins la vie de votre batterie sera importante (même si le fabricant précise un courant de charge de 3C). Si rien n'est spécifié sur la batterie ou par le fabricant, en cas de moindre doute ou simplement pour allonger la vie de votre batterie : chargez à 1C maximum !

Pour charger une batterie LiPo, un chargeur va fournir un courant constant correspondant au courant de charge spécifié par le constructeur (ou 1C). Pour maintenir ce courant, la tension va alors progressivement augmenter au fil de la charge, jusqu'à atteindre la tension de charge de 4,2 volts par accu. À ce moment, du courant circule toujours du chargeur vers la batterie qui n'est pas pleinement chargée. Le chargeur va alors changer de mode et passer en tension constante et ne plus réguler le courant. Durant cette phase finale, le courant va alors progressivement baisser jusqu'à arriver à une valeur limite qui marque la fin de la charge.



Le LiPo Rider Pro seul ne sert à rien. Il devra au minimum être accompagné d'un accumulateur LiPo de la capacité adaptée à votre projet. Ceci vous coûtera une grosse poignée d'euros, mais tout l'aspect « autonomie » de votre projet sera achevé d'un coup. Il ne vous restera qu'à vous occuper du reste.

Vous l'avez compris, non seulement charger une batterie ne se résume pas simplement à fournir une alimentation et attendre que cela se passe, mais en plus il faut prendre en compte la capacité de la batterie, le nombre de cellules et la façon dont elles sont connectées dans la batterie. De plus, il arrive que toutes les cellules ne se chargent pas de façon équivalente. On parle alors d'équilibrage de charge, et il faut mesurer les tensions aux pôles de chaque cellule durant le processus. Et enfin, par mesure de sécurité, il faut également mesurer la température de la batterie ou des éléments afin de stopper la charge en cas de problème.

Charger une batterie LiPo, même 1S (donc un accu) n'est pas de tout repos et ne s'improvise pas. C'est pour cette raison qu'il existe des circuits intégrés spécialisés, configurés pour un type spécifique de batterie. Votre smartphone ou même votre vieux téléphone mobile en contient un. Les chargeurs de LiPo, comme ceux destinés au modélisme en font également usage, mais ils intègrent en plus un circuit complet permettant une configuration afin d'adapter la charge à la capacité de la batterie, au taux de charge souhaité et au nombre de cellules en présence.

Pour en revenir maintenant à l'aspect solaire, on distingue un peu mieux le problème. Interrompre la charge d'une batterie LiPo n'est pas un problème, tout se passera ensuite comme s'il s'agissait d'une nouvelle charge moins longue. Il faut en revanche que, selon le moment dans le cycle de charge, le courant et la tension puissent être fournis. Si, par temps couvert, votre panneau de cellules photovoltaïques n'est pas en mesure de fournir l'énergie adéquate, il ne faut pas tenter la charge avec de mauvaises valeurs. Ceci est également valable par temps clair, avec un nuage ou un objet quelconque, bloquant les rayons du soleil.

Le chargeur solaire doit donc non seulement contrôler l'énergie fournie, mais également celle obtenue. Tout se complique...

Ce n'est pas tout. Sans doute n'aurez-vous pas envie de brancher/débrancher la batterie à chaque utilisation, mais préféreriez connecter directement votre projet sur le trio chargeur/panneau/batterie et faire en sorte, par la même occasion, que la batterie ne se décharge que lorsque c'est nécessaire. Seulement voilà, le soleil n'en fait qu'à sa tête et la batterie ne fournit pas de courant en 5V ou en 3,3V. Cela se complique vraiment...

Il y a des domaines où il est plaisant de tout construire, mais lorsqu'il est question de risque d'explosion, il faut se montrer prudent et ici cela signifie étudier

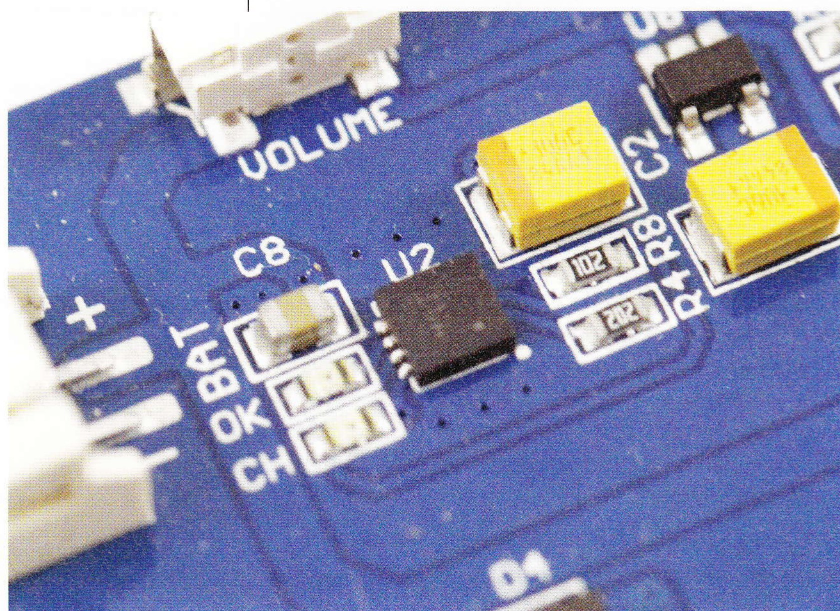
la question à fond, ce qui demande du temps, des expérimentations et beaucoup d'efforts. Confectionner un système de charge solaire pour une batterie ou un simple accu n'est pas chose impossible, mais ce n'est pas quelque chose qui s'improvise. Si l'objectif premier est de rendre autonome ou nomade un projet, il est tantôt préférable de se tourner vers une solution existante pour concentrer ses efforts sur le cœur de son projet.

2. LIPO RIDER PRO

Il existe un produit qui, lorsqu'il est question de contrôle de charge et de l'utilisation d'une source solaire, revient fréquemment sur les forums et dans les discussions comme la solution la plus accessible : le LiPo Rider.

Ce circuit développé et vendu par *Seeed Studio* est relativement simple, peu coûteux (~14€) et surtout en open hardware. Ceci signifie que sa conception, son schéma ainsi que la liste des composants utilisés, sont distribués à la façon d'un logiciel libre, sous une licence permettant à quiconque de reproduire, faire évoluer et distribuer des versions de ce même circuit. Le LiPo Rider est donc un peu plus qu'un simple « produit », il est ouvert.

Le circuit intégré CN3065 de Consonance est le cœur du LiPo Rider Pro. C'est ce composant qui a pour travail de contrôler la charge de l'accumulateur dans des conditions optimales.



Il existe deux déclinaisons de ce circuit, le LiPo Rider (v0.9b/V1.0/V1.1) et le LiPo Rider Pro (v0.9/v1.0/v1.2). Les deux modèles se distinguent par les composants utilisés ainsi que, aussi et surtout, par le courant maximum qui peut être fourni à un montage externe. En dehors de ces points, la logique des circuits est identique et se résume assez facilement aux connecteurs disponibles :

- un connecteur mini USB femelle permettant de connecter une source d'alimentation en 5V (bloc d'alimentation ou PC/Mac) ;
- un connecteur (JST 2.0 sur le modèle v1.2 testé) pour panneau solaire 4,8V-6V (maximum 6,5V pour 10s) de 3W par exemple, comme celui vendu également par Seeed Studio à un peu plus de 11€ (138x160mm monocristallin) ;
- un connecteur (également JST 2.0) pour accumulateur LiPo (« accumulateur », pas « batterie ». Il y a une seule cellule). La capacité de l'accu n'est pas critique. Plus elle sera importante, plus la charge durera longtemps en fonction des conditions d'ensoleillement ;
- un connecteur USB A femelle pour brancher le montage ou la carte à alimenter.

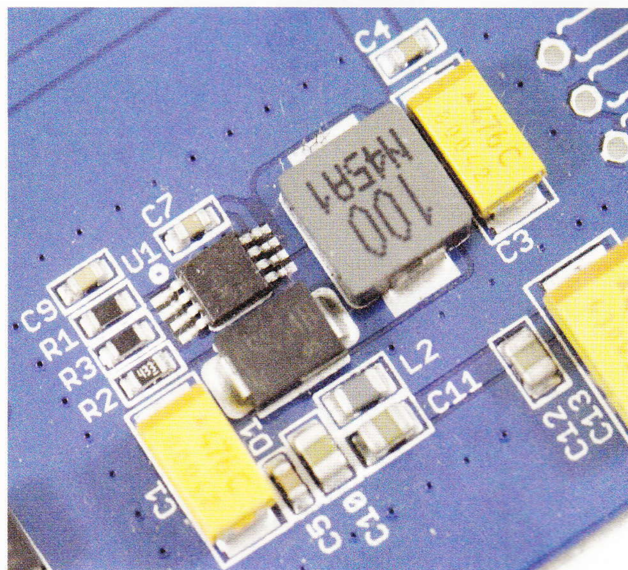
L'accu sera chargé soit via l'alimentation en mini USB, soit par le panneau solaire. L'ensemble peut donc fonctionner sans panneau à la façon d'un

périphérique mobile comme un smartphone. Cette connexion USB permettra également de laisser communiquer une carte comme Arduino avec un système hôte comme un PC (moniteur série) tout en profitant de la connexion pour charger l'accu. Bien entendu, lorsque l'alimentation mini USB est connectée au LiPo Rider, c'est elle qui fournira le courant au montage branché sur la prise USB A.

L'ajout d'un panneau solaire permettra alors le rechargement de l'accumulateur par ce biais, mais, par contre, ne permettra l'alimentation de votre montage qu'en fonction de l'énergie disponible dans l'accu. Il n'est donc pas possible d'alimenter un montage uniquement par l'énergie solaire, en l'absence d'accu. Sans alimentation mini USB, c'est toujours l'accu qui fournit le courant.

Le LiPo Rider Pro, capable de fournir jusqu'à un ampère, peut être désormais vu comme le successeur du LiPo Rider qui n'est plus commercialisé directement par Seeed Studio. Cet ancien modèle peut toujours être trouvé auprès d'autres fabricants à moins de 10€, mais il est préférable de l'éviter, même s'il est moins coûteux (vous serez rapidement limités par les 350 mA maximum qu'il peut fournir).

La construction du LiPo Rider Pro repose sur trois circuits intégrés, dont deux sont cruciaux. Nous avons d'une part le contrôleur de charge pour accumulateur CN3065 fabriqué par *Consonance*. Celui-ci a pour tâche de charger l'accu via le processus



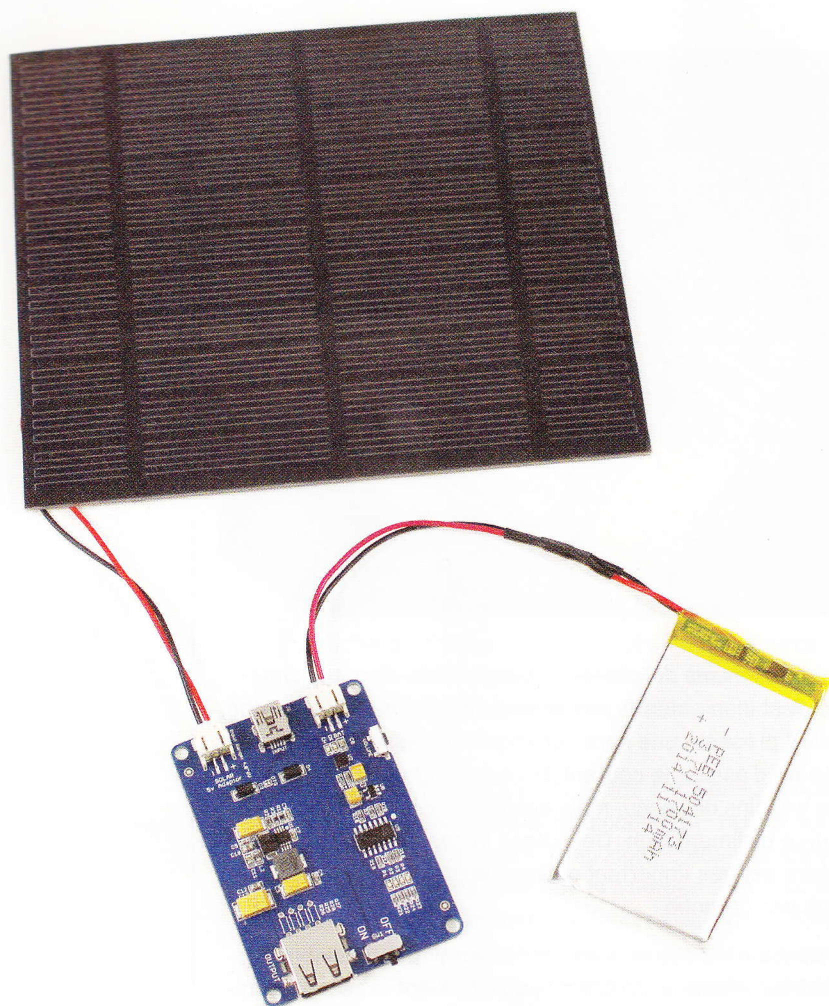
Cette partie du LiPo Rider Pro est le circuit chargé de transformer la tension fournie par l'accumulateur en 5 volts pouvant être utilisés par des cartes comme Arduino via le port USB A femelle.

que nous avons détaillé en début d'article. Ce circuit intégré n'est bien entendu pas le seul disponible sur le marché, mais il présente une caractéristique intéressante : il est en mesure d'ajuster le courant de charge non seulement selon des besoins de l'accu, mais également en fonction de la source d'alimentation. Ceci, dans le cas d'un panneau solaire est très important puisque la stabilité de la source n'est pas garantie.

L'accu n'est pas en mesure de fournir 5V, mais uniquement, au mieux, 3,7V. C'est là qu'intervient le second composant le plus important, l'ISL97516 d'Intersil. Il s'agit d'un convertisseur boost, chargé d'utiliser une tension entre 2,3V et 5,5V et de produire un courant avec une tension déterminée par la configuration d'une de ses broches (5V dans ce cas). L'efficacité ici est la clé puisqu'il n'est pas question de perdre inutilement le moindre milliwatt, or justement ce circuit intégré est spécialement destiné à ce type d'application (efficacité >90%).

Enfin, nous avons un quadruple amplificateur opérationnel (op-amp) STMicroelectronics LMV324ID, couplé à un régulateur Micrel MIC5205, le tout connecté à l'anode (+) de l'accu via un bouton poussoir. Le régulateur permet d'obtenir une tension de 2,5V et celle-ci est comparée, via les op-amp, aux tensions présentes aux broches d'une série de résistances connectées, via le bouton, entre l'accu et la masse. Les sorties des op-amp sont reliées à des leds et des résistances.

Ainsi, une pression sur le bouton va connecter la série de résistances à la masse et la tension à chacune des interconnexions sera comparée aux 2,5V par les op-amp pour éventuellement allumer 0, 1, 2, 3 ou 4 leds. Celles-ci



Agencement typique du LiPo Rider Pro avec en haut un panneau solaire 3W et sur la droite un accumulateur LiPo de 1700 mAh. En choisissant judicieusement le panneau et l'accumulateur, et en travaillant énormément sur la consommation de votre projet, ce type de configuration peut vous assurer un chargement solaire en journée et un fonctionnement sur accumulateur la nuit, et donc un fonctionnement 24h/24 7j/7.

permettent donc d'avoir une approximation de l'état de la charge de l'accu : 0-10%, 10-30%, 30-60%, 60-90% et 90-100%. On regrettera qu'un connecteur ne soit pas disponible afin de « remonter » cette tension au montage connecté au LiPo Rider Pro afin d'obtenir un niveau de charge via un convertisseur analogique/numérique (broches A0 à A5 d'une Arduino UNO par exemple).

3. LE CÔTÉ PRATIQUE ET FINANCIER

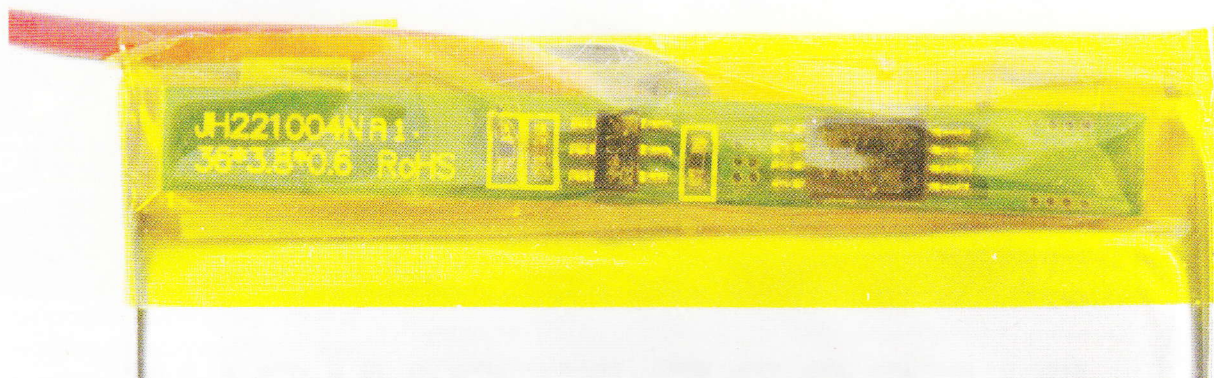
Éloignons-nous un instant des aspects purement techniques pour répondre à une question plus terre-à-terre du type « *ok, rendre un projet Arduino autonome me coûtera combien ?* ».

Pour atteindre votre objectif, il vous faudra avant tout connaître la consommation de votre montage final, même approximative. Ceci vous permettra de déterminer la capacité nécessaire pour l'accu (en comptant une marge d'erreur) et donc la taille/puissance du panneau solaire destiné à recharger l'accu. Un panneau de 138x160mm de 3 watts fournira par exemple idéalement 540 mA en 5,5V. En ne comptant aucune perte et avec un ensoleillement parfait, il faudra donc plus de 3h pour charger entièrement un accu de 1700 mAh. Dans le cas d'un accu de 7700 mAh on arrive, toujours dans des conditions idéales, à plus de 14 heures ! Je ne sais pas pour vous, mais ici en Alsace, même en été et en rase campagne, c'est impossible en pratique. Dépenser quelques 40€ d'accu LiPo de cette taille pour un projet solaire n'a donc pas de sens.

Pour une configuration standard, l'idée générale est donc d'avoir un montage qui consomme moins d'énergie en l'absence de soleil qu'il n'en faut pour recharger l'accu en journée. Il ne s'agit donc pas d'assembler n'importe quel projet, mais partir dès le début avec en tête une seule chose : réduire la consommation d'énergie au maximum.

Quoi qu'il en soit, voici l'une de mes configurations à base de LiPoRider Pro :

- un LiPo Rider Pro avec panneau 3W : 22,05 euros chez Seeed Studio ;



- 10 connecteurs JST 2.0 : 0,88 euros (souvent nécessaires en achetant des accus ailleurs tantôt équipés de connecteurs JST 2.54) ;
- 1 accu LiPo 1700 mAh provenant d'un vendeur eBay allemand (kt-elektronic) : 11,30 euros + 2,5 euros de port (il est possible d'acheter des accus LiPo directement chez Seeed Studio, mais la livraison se fait alors par transporteur et est hors de prix alors qu'elle est offerte par défaut).

Budget total pour l'ensemble : 36,73 euros.

Ceci n'est pas une petite somme, mais mieux vaut se montrer prudent. Vous trouverez sur eBay tout un tas de vendeurs proposant par exemple des panneaux solaires de toutes sortes et de tous modèles. Les spécifications annoncées ne sont souvent pas réelles, sauf à tomber sur un vendeur prenant soin de sa réputation. Le point sensible est en particulier le type de cellule photovoltaïque : monocristalline (meilleur rendement, plus cher) ou polycristalline (rendement faible, coût réduit). Vous comprendrez donc que certains soient tentés de vendre du polycristallin comme étant du monocristallin...

L'autre élément important est l'accu lui-même. Là encore, il faut faire très attention, car les capacités annoncées peuvent parfois être fausses. Pire encore, certains n'hésitent pas à vendre des éléments recyclés, trafiqués ou endommagés (voire remplis de sable ou de terre). Dans ce cas, il est souvent de bon ton de préférer un vendeur à proximité (en Europe au minimum), ayant d'excellentes évaluations et commentaires. Enfin, achetez TOUJOURS des accus intégrant un circuit de protection et dont les photos de produits montrent le circuit de protection (même si ce n'est pas une preuve). N'achetez JAMAIS de cellules LiPo sans circuit (dites « raw » ou « bulk ») ou avec un circuit non visible. Cet élément est présent pour éviter la surcharge, la décharge excessive, les surtensions, les courts-circuits, etc. Ceci n'est pas une protection absolue, mais c'est un minimum indispensable.

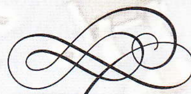
4. POUR FINIR

Le Lipo Rider Pro est, à mon goût, une solution rapide et efficace pour rendre un projet énergiquement autonome. On règle ainsi, pour moins de 40 euros, l'aspect purement matériel et on peut alors se pencher sur le cœur du problème. Il ne suffit pas, en effet, d'avoir sous le coude quelques 1700 mAh, encore faut-il les utiliser judicieusement. Et c'est précisément là qu'il faudra investir son énergie et son temps afin de trouver et d'utiliser les moindres petites astuces qui permettront de grappiller de-ci de-là quelques milliampères, et donc augmenter l'autonomie de votre projet. Faites également très attention aux conditions environnementales de votre installation. Les LiPo sont sensibles aux chocs, aux températures élevées et au gel. Restez prudent. DB

Les accumulateurs LiPo doivent impérativement être équipés de circuit de protection garantissant un certain niveau de sécurité. Ceci ne vous met pas à l'abri de toutes les bêtises possibles, mais constitue un niveau minimum de protection.

DU CODE ATOMIQUE DANS VOS CROQUIS ARDUINO ?

Denis Bodor



Non, il n'est pas question de radiations ou de champignons nucléaires dans les croquis, encore moins de mutation du code pour obtenir un super-Arduino. Dans un contexte de programmation, le mot « atomique » est à prendre au sens de sa racine grecque « atomos », signifiant « qu'on ne peut diviser ». Une opération atomique est donc une opération qui s'exécute entièrement sans pouvoir être interrompue. Pourquoi ceci est important ? Explications...

La question de l'atomicité du code ne se pose qu'à une condition : que son exécution puisse être interrompue. En effet, avec un croquis parfaitement linéaire simplement composé d'une succession d'instructions, de tests et de boucles, il n'existe aucune situation où l'exécution normale d'un code peut être stoppée pour en exécuter un autre. Avec une condition ou un appel de fonction, rien n'est interrompu, le code suit simplement son cours.

L'Arduino, son « langage », ses bibliothèques et la plateforme matérielle elle-même, permettent cependant d'utiliser la notion d'interruption consistant justement à utiliser cette technique. Un exemple courant consiste au cas de figure hypothétique suivant : votre croquis fait clignoter une led et ce clignotement doit être stoppé lorsqu'un bouton poussoir est enfoncé.

Il y a deux manières d'envisager la chose :

- Votre croquis teste le bouton poussoir dans la boucle principale avant chaque changement d'état de la led. Cette technique appelée *polling* (littéralement « sondage » ou « scrutation ») est une **attente active**, le code, de lui-même, agit pour vérifier l'état d'une entrée (par exemple).
- Utiliser une interruption. La boucle principale se contente de changer l'état de la led de façon répétée, mais son exécution est interrompue lors d'un événement configuré à l'avance, comme le changement d'état sur une entrée de la carte. Il n'y a rien dans la boucle elle-même qui teste l'entrée (bouton poussoir). L'exécution de la boucle est interrompue par... une interruption.

La seconde solution est souvent celle à privilégier pour un grand nombre de croquis, en particulier ceux reposant sur une intervention extérieure, qu'il s'agisse d'un bouton poussoir ou un signal reçu par la carte. Bien entendu, si la tâche principale du croquis consiste précisément à scruter un tel changement, l'utilisation du *polling* n'est pas un problème, mais ceci est rarement le cas. Le plus souvent, le projet consiste en une tâche récurrente ou courante (rafraîchir un écran, contrôler des leds ou un moteur, relever une valeur sur un capteur, etc.) qui est l'objet même de la boucle principale et un événement ponctuel à traiter survient.

Notez d'ailleurs que bon nombre de modules disposent d'une broche « INT » (*interrupt*) ou « IRQ » (*Interrupt ReQuest*) leur permettant justement d'interrompre l'exécution du code d'une carte comme Arduino et ainsi signifier qu'il se passe quelque chose ou qu'ils ont des données à fournir.

Mais le plus simple est encore de découvrir le monde des interruptions par l'exemple mais, pour que le plus grand nombre puisse le faire, sans utilisation de matériel externe particulier, nous utiliserons une autre source d'interruption : un *timer*.

1. EXÉCUTER UN CODE À INTERVALLE RÉGULIER : TIMER

Le microcontrôleur d'une carte Arduino n'est pas juste un processeur et de la mémoire (SRAM et flash). Le composant intègre également des périphériques : des compteurs, des comparateurs, des convertisseurs analogique/numérique et des *timers* (« temporisateur » ? « minuterie » ? Tout le monde dit « timer » de toute façon).

L'Atmel ATmega328P d'une carte Arduino UNO, par exemple, possède deux timers/compteurs de 8 bits et un de 16 bits. Il s'agit de périphériques, ce qui signifie qu'ils fonctionnent indépendamment du code exécuté par le processeur. Les timers permettent énormément de choses : on règle leur vitesse et différents paramètres de fonctionnement et ils travaillent tout seuls. Il est même possible de les configurer de façon à piloter directement une ou plusieurs sorties

de la carte. Ce sont eux qui permettent l'utilisation de la fonction **analogWrite()** pour contrôler par exemple l'intensité lumineuse d'une led ou la vitesse d'un moteur.

Les timers sont donc déjà utilisés en coulisse sans que vous le sachiez. Il est également possible d'en prendre le contrôle en manipulant directement les registres du microcontrôleur, mais il y a plus simple. Il existe en effet une bibliothèque, disponible via le gestionnaire de bibliothèque de l'environnement Arduino, permettant d'utiliser simplement un timer : TimerOne.

Une fois cette bibliothèque installée, son utilisation ne présente pas vraiment de difficulté :

```
#include <TimerOne.h>

// variable pour l'état de la led
int etatLED = LOW;

void setup() {
  // Led sur la carte en sortie
  pinMode(LED_BUILTIN, OUTPUT);
  // Configuration du timer
  // un déclenchement toutes les 100000
microsecondes
  Timer1.initialize(100000);
  // Lors d'un déclenchement une interruption est
  // générée et cette fonction est appelée
  Timer1.attachInterrupt(cliLED);
}

// fonction appelée lors de l'interruption
void cliLED() {
  // On inverse l'état
  // Si LOW alors HIGH, si HIGH alors LOW
  etatLED = !etatLED;
  // changement d'état de la led
  digitalWrite(LED_BUILTIN, etatLED);
}

void loop() {
  // rien ici
}
```

Ceci est totalement représentatif de la notion d'interruption. Notre boucle principale est vide, tout se passe dans la fonction **cliLED()** appelée lors d'une interruption, elle-même déclenchée par le timer, configuré dans **setup()**. Dès l'appel de **Timer1.initialize()** avec en argument un nombre de microsecondes (1/1000000 de seconde), le timer est initialisé et l'appel à **Timer1.attachInterrupt()** configure le déclenchement d'interruption et l'appel à notre fonction **cliLED()**. **cliLED()** sera alors automatiquement appelé toutes les 100000 microsecondes.

Nous voyons ici une troisième façon de faire clignoter la led d'une carte Arduino, s'ajoutant aux exemples intégrés que sont la simple boucle utilisant **delay()** (**Blink**) et l'utilisation de **millis()** (**BlinkWithoutDelay**).

À propos de ces fonctions, il est important de noter que les trois timers d'un ATmega328P sont utilisés par les bibliothèques Arduino. Le *Timer0* (8 bits) est utilisé par les fonctions **delay()** et **millis()**, le *Timer2* est utilisé par la fonction **tone()** pour générer un son, et *Timer1* (16 bits) est celui en œuvre pour la bibliothèque *Servo*.

analogWrite() par contre peut utiliser les trois timers en fonction de la broche contrôlée. Le fait de mettre en œuvre la bibliothèque *TimerOne* nous interdira l'utilisation de cette fonction sur les sorties 9 et 10 sur Arduino Uno, 9 à 11 sur Leonardo ou encore 11 à 13 sur Arduino Mega. On pourra en revanche utiliser la fonction **Timer1.pwm()** pour utiliser ces sorties ,mais en passant une valeur entre 0 et 1023 (et non entre 0 et 255 comme avec **analogWrite()**).

2. TIMER, INTERRUPTION ET MAUVAIS CODE

Nous savons maintenant configurer un timer, déclencher une interruption et associer une fonction à cette dernière. Il est temps de pousser un cran plus loin et

d'aborder le problème d'interaction entre la boucle principale et la routine d'interruption. Nous allons reprendre le croquis précédent et ajouter une petite fonctionnalité :

```
#include <TimerOne.h>

int etatLED = LOW;
volatile unsigned int compteur = 42000;

void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
  // Nouvelle broche en sortie
  pinMode(4, OUTPUT);
  Timer1.initialize(100000);
  Timer1.attachInterrupt(cliLED);
}

void cliLED() {
  etatLED = !etatLED;
  digitalWrite(LED_BUILTIN, etatLED);
  // on décrémente le compteur 16 bits
  compteur--;
}

void loop() {
  // on teste la valeur du compteur
  if(compteur == 0)
    digitalWrite(4, HIGH);
}
```

Tout d'abord, nous avons une nouvelle sortie de la carte Arduino qui est utilisée. Qu'elle soit connectée en réalité à une résistance et une led n'a pas réellement d'importance ici. C'est une simple action que nous conditionnons en fonction de la valeur d'une variable. La variable en question est `compteur`, un entier sur 16 bits, initialisé à 42000 et décrémenté par notre routine déclenchée par l'interruption générée par le timer.

Le problème ici n'est pas immédiatement visible et est en rapport avec le fonctionnement même du microcontrôleur équipant la majorité des cartes Arduino : un Atmel AVR 8 bits. Le processeur intégré dans le microcontrôleur travaille en 8 bits, mais nous testons une variable non signée de 16 bits ou deux octets (un `int`). Ceci ne peut donc se faire qu'en deux opérations : tester les 8 premiers bits (ou 8 bits de poids faible), puis tester les 8 autres bits (8 bits de poids fort).

Il existe une probabilité que le phénomène suivant survienne :

- la valeur du compteur est par exemple `0x0100`,
- le processeur teste les 8 premiers bits, soit `0x00`,
- une interruption survient et `cliLED()` est exécuté,
- notre variable `compteur` passe de `0x0100` à `0x00FF`,
- l'exécution normale reprend suite à l'interruption,
- le processeur teste les 8 bits restants qui sont maintenant `0x00` et non plus `0x01`,
- le test retourne VRAI, car les deux groupes de 8 bits ont effectivement été testés à zéro,
- l'action est déclenchée et la broche 4 change d'état alors qu'il restait 255 décrémentations à opérer.

Ce type de problème est très difficile à détecter, en particulier dans un croquis de grande taille faisant massivement usage d'interruptions et de variables de taille importante (`int`, `long`, `double`, etc.). Tout dépendra du moment exact où l'interruption surviendra.

De plus, ceci peut souvent découler sur ce que les programmeurs appellent un heisenbug : un bogue qui subitement n'apparaît plus dès qu'on cherche à le détecter. En effet, pour comprendre ce qui se passe, on ajoutera des tests ou des sorties sur le moniteur série. Cela aura pour effet de changer le comportement du croquis, les conditions du test ou encore la chronologie des exécutions, et ceci peut alors faire fonctionner le programme sans problème. Le bogue est toujours là et en retirant le code destiné à l'analyser, il refera son apparition... ou pas.

3. LA SOLUTION : UNE OPÉRATION ATOMIQUE

Pour éviter le problème directement lié à l'interruption de l'évaluation de notre compteur, la solution est simplissime : il suffit de ne pas permettre d'interruption au moment du test.

Dans les faits malheureusement, les choses ne sont pas si simples, car la manière de gérer cette évaluation dans un **if**, un **for** ou un **while** n'est pas notre affaire, mais celle du compilateur. La seule chose que nous pourrions faire à ce niveau est d'interdire les interruptions dans toute la portée de la condition, ce qui contredit l'intérêt même du mécanisme d'interruption.

Mais en y regardant de plus près, le test n'est pas la source du problème, c'est la variable elle-même. Nous pouvons envisager les choses autrement : disposer d'une variable dans notre boucle principale et y copier la valeur du compteur. C'est alors cette copie qui se fera sans interruption et le test suivra sur la variable locale alors que les éventuelles interruptions continueront d'être traitées. La copie de la variable sera donc notre opération atomique.

Il existe plusieurs façons de désactiver temporairement les interruptions pour rendre une opération atomique. Le code que l'on retrouve généralement dans les croquis Arduino ressemble à :

```
// désactivation interruption
noInterrupts()

// Mon opération atomique
compteurCopie = compteur;

// activation des interruptions
interrupts()
```

Ce sont d'ailleurs ces lignes que l'on retrouve dans l'exemple **Interrupt** livré avec la bibliothèque TimerOne. Mais les solutions populaires ne sont pas toujours les meilleures. Ces fonctions, **interrupts()** et **noInterrupts()**, ne sont que des macros définies dans **Arduino.h** :

```
#define interrupts() sei()
#define noInterrupts() cli()
```

sei() et **cli()** proviennent de l'avr-libc, le socle sur lequel repose l'ensemble des bibliothèques standards Arduino. Mais lorsqu'on regarde leur documentation, il est dit « Pour implémenter l'accès atomique à des objets multi-octets, il est préférable d'utiliser les macros **util/atomic.h**, plutôt que de le faire manuellement avec **cli()** et **sei()** ». En d'autres termes **interrupts()** et **noInterrupts()**, simples synonymes de **sei()** et **cli()** ne représentent pas la voie à suivre.

util/atomic.h fournit en revanche quelque chose de bien plus intéressant. Voici notre croquis mis à jour dans ce sens :

```
#include <TimerOne.h>
// ajouts des fonctions utilitaires
// avr-libc
#include <util/atomic.h>

int etatLED = LOW;
volatile unsigned int compteur = 42000;

void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
  pinMode(4, OUTPUT);
  Timer1.initialize(100000);
  Timer1.attachInterrupt(cliLED);
}

void cliLED() {
  etatLED = !etatLED;
  digitalWrite(LED_BUILTIN, etatLED);
  compteur--;
}

void loop() {
  // une nouvelle variable pour la copie
  // du compteur
  unsigned int compteurCopie;

  ATOMIC_BLOCK(ATOMIC_RESTORESTATE) {
    // Tout ce qui se trouve dans la
    // portée de ce bloc ne peut être
    // interrompu
    // On copie la valeur du compteur
    compteurCopie = compteur;
  }

  // on teste la copie
  if(compteurCopie == 0)
    digitalWrite(4, HIGH);
}
```

Un `#include` nous permet de disposer de la fameuse macro recommandée. Celle-ci est utilisée un peu plus bas lors de la copie :

```
ATOMIC_BLOCK(ATOMIC_RESTORESTATE) {
    compteurCopie = compteur;
}
```

L'utilisation est bien plus intuitive puisque `ATOMIC_BLOCK()` prend en argument un type qui peut être :

- `ATOMIC_RESTORESTATE` : l'atomicité du code qui suit entre `{` et `}` est garantie et la configuration est restaurée en sortie ;
- `ATOMIC_FORCEON` : réactive les interruptions en sortie de bloc, quelle que soit la situation avant la désactivation. Ceci permet d'économiser un peu de mémoire puisque la macro n'a pas besoin de sauvegarder l'état avant la désactivation.

On préférera généralement la première option à moins d'avoir vraiment besoin de grappiller un peu de mémoire.

Mais le principal avantage de cette macro par rapport aux deux fonctions vues précédemment est la syntaxe elle-même. L'utilisation d'une portée entre accolades est plus élégante et surtout plus sûre. Oublier un `interrupts()` est facile et invisible, oublier le `}` provoquera une erreur à la compilation. La lisibilité est également plus importante, le code atomique est simplement celui dans la portée de la macro.

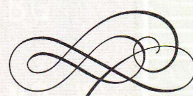
4. LE MOT DE LA FIN

Il ne faut pas perdre de vue que le « langage » Arduino est avant tout du C/C++ accompagné de « facilités ». `interrupts()` et `noInterrupts()` ont été ajoutés, car `sei()` et `cli()` ne sont pas des noms de fonction très explicites (en fait il s'agit du nom des instructions assembleur équivalentes). Ceci permet donc à l'utilisateur et au programmeur débutants d'associer rapidement un concept à une fonction.

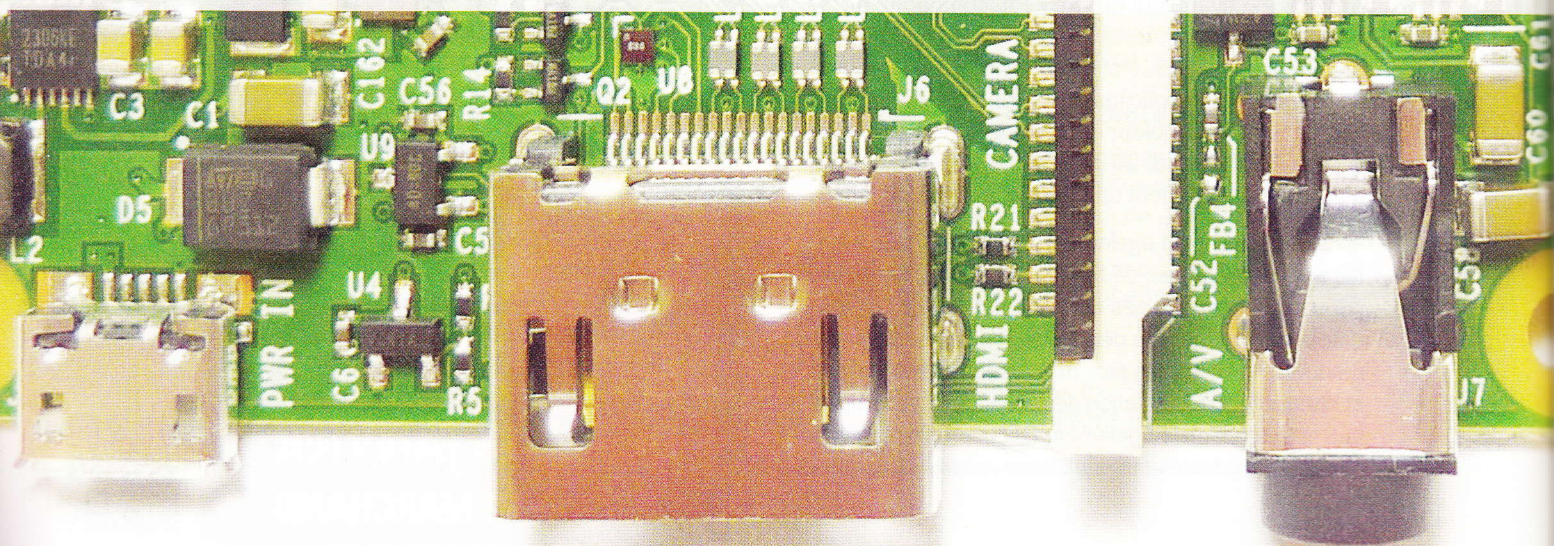
Cependant, il est également important, après avoir fait ses premiers pas, de regarder cela avec un œil plus critique et surtout ne pas s'en tenir aux facilités mises à disposition. Une telle démarche vous conduira forcément vers les couches plus basses de l'environnement et donc à la documentation de l'avr-libc ou du compilateur lui-même, mais également aux manuels du microcontrôleur (*datasheet* et *application notes*). DB

COMPILEZ UN NOUVEAU NOYAU POUR VOTRE RASPBERRY PI

Denis Bodor



Cet article est destiné aux utilisateurs avancés du système Raspbian, permettant d'utiliser les cartes Raspberry Pi. La compilation et l'installation d'un nouveau noyau sont des manipulations qui ne sont généralement pas nécessaires, sauf cas très particuliers. Si vous cherchez à faire une telle chose, vous savez en principe pourquoi, en quoi consiste l'opération et quels sont les risques.



Depuis des années, quel que soit le système GNU/Linux utilisé, qu'il s'agisse de PC, de machines anciennes ou exotiques ou de nano-ordinateurs, tout est fait, pour que l'utilisateur n'ait pas à se lancer dans les péripéties d'une compilation de noyau. Le principe de « distribution » GNU/Linux même est presque entièrement axé vers ce type de simplifications et de facilités. Le noyau, les bibliothèques et les applications sont rendus disponibles et installables aisément pour tous, ou presque.

Il n'en a pas toujours été ainsi. En effet, aux premières heures de l'aventure GNU/Linux, la (re)compilation était une activité courante sinon indispensable, et ce, aussi bien pour les applications que pour le noyau. Les raisons pour lesquelles on préfère utiliser les sources et compiler soi-même ses programmes étaient et sont relativement simples : disposer d'une version la plus récente possible, étudier le code, tester des modifications ou des corrections ou tout simplement profiter de fonctionnalités qui ne sont pas encore présentes dans la version disponible. Ces raisons sont toujours valables aujourd'hui, mais les choses ont un peu changé.

La plupart des projets de logiciels libres utilisent une stratégie de développement accélérant énormément la mise à disposition des dernières versions stables, en test ou non stables. Les distributions font de même en diffusant régulièrement des mises à jour et/ou de nouvelles versions majeures. Enfin, avec des projets de développement comme Raspbian, et en particulier le noyau incluant les spécificités pour les cartes Raspberry Pi, les mises à jour sont intégrées avec celles du reste du système. C'est d'ailleurs ainsi que tout s'est déroulé sans le moindre problème avec l'arrivée des caméras Pi, de la Raspberry Pi 2 ou du très récent écran 7 pouces officiel. Avec un système à jour, aucune manipulation n'a été nécessaire et un simple redémarrage était suffisant.

Il ne nous reste donc que deux principales raisons : comprendre comment tout cela fonctionne et tenter de faire fonctionner un périphérique un peu trop exotique pour être supporté par défaut. Ces motivations supposent que vous disposez déjà de certaines compétences techniques que je pourrai qualifier d'avancées. Dans les paragraphes qui suivent, je me contenterai donc, pour une fois, de décrire le processus sans pour autant expliquer outre mesure des concepts et des commandes que j'estime déjà connus.

1. NÉCESSITÉ DE LA COMPILATION CROISÉE

Il n'est pas question de compiler un noyau directement sur la carte Raspberry Pi elle-même. Même avec un modèle 2B (4 cœurs et 1 Go de mémoire) une telle manipulation est certes possible, mais stupide. Il faut ici entrer dans une logique de système embarqué : la plateforme est censée exécuter des programmes, non servir de plateforme de développement système.

La compilation croisée consiste, depuis une plateforme donnée, à compiler des codes destinés à fonctionner sur une autre. Dans notre cas, ceci revient à compiler sur PC (architecture x86 ou amd64) un noyau qui fonctionnera sur la Raspberry Pi. Pour que cela soit possible, nous ne pouvons pas utiliser le compilateur présent par défaut sur le PC (l'utilisation de GNU/Linux est ici implicite). Celui-ci, en effet, est conçu pour produire des programmes (binaires) qui fonctionnent sur la même plateforme. Ce dont nous avons besoin est une chaîne de compilation croisée.

Pour l'installation, sur Ubuntu, nous utiliserons simplement la commande :

```
$ sudo apt-get install gcc-arm-linux-gnueabihf
```

et sur Debian :

```
$ sudo dpkg --add-architecture armhf
$ sudo apt-get update
$ sudo apt-get install gcc-arm-linux-gnueabihf
```

À ce stade et en raison des nombreuses dépendances de ce paquet, vous disposerez alors d'une chaîne de compilation (compilateur C, assembleur, éditeur de liens, etc.) complète et adaptée pour la production de binaires ARM. Voici votre boîte à outils pleine et prête à l'emploi.

2. ALLONS-Y !*

2.1 Récupération des sources

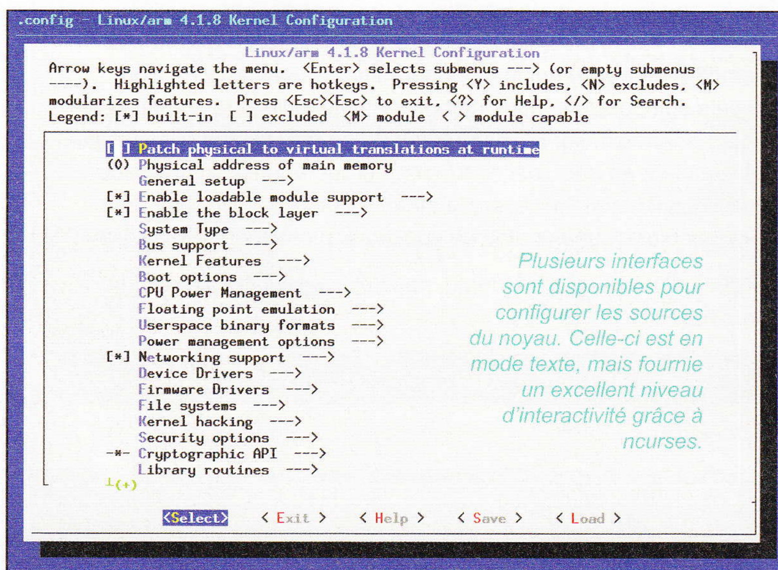
Nous ne pouvons pas utiliser ici un noyau Linux standard, car certaines spécificités ne sont pas prises en charge par le noyau officiel pour l'instant. La fondation Raspberry Pi tient donc à jour sa propre version de développement, mise à disposition sur GitHub. Il est également très important d'utiliser cette version, car si vous corrigez un problème, il vous faudra alors utiliser le système de *fork* et de *pull request* de GitHub pour contribuer.

Pour récupérer les sources, il suffit d'utiliser Git en clonant le dépôt officiel sur votre machine :

```
$ git clone https://github.com/raspberrypi/linux.git
```

2.2 Configuration

Les sources d'un noyau Linux sont configurables. Comprenez par là qu'il ne s'agit pas d'une simple application générique avec une liste fixe de fonctionnalités, mais d'une masse vraiment énorme de code, organisée en systèmes et sous-systèmes. Il faut choisir avec précision quelles fonctionnalités devront être activées ou non.



Plusieurs interfaces sont disponibles pour configurer les sources du noyau. Celle-ci est en mode texte, mais fournie un excellent niveau d'interactivité grâce à ncurses.

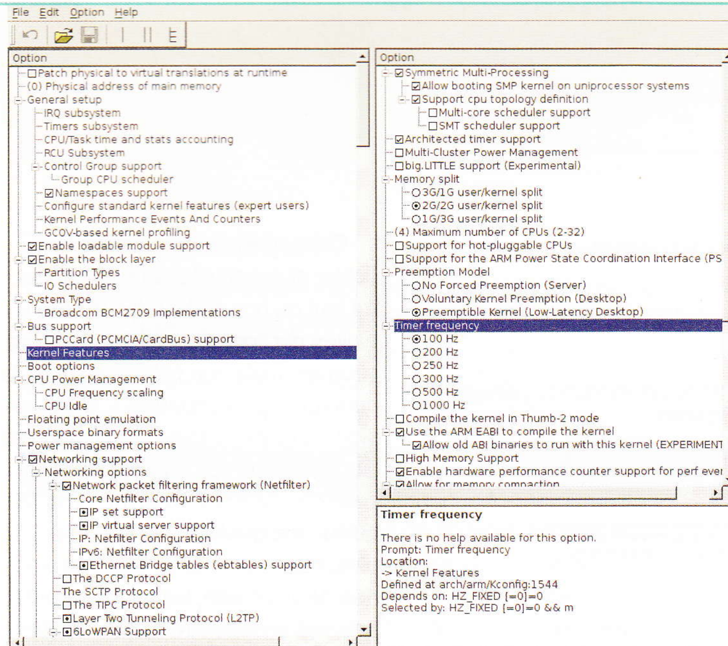
Pour simplifier les choses, les sources du noyau intègrent un système de configuration se présentant sous forme d'une interface avec menus, options et listes déroulantes. Tout ceci permet de composer les caractéristiques du noyau souhaité ainsi que le matériel qu'il devra supporter, que ce soit en interne ou sous forme de modules utilisables avec `modprobe/insmod/rmmod` ou le *Device Tree* (voir article sur le double écran SPI dans ce numéro).

Cependant, comme ces choix et configurations peuvent s'avérer très denses et complexes, il est d'usage de reposer sur ces fichiers de configuration. Ainsi, avec les sources du noyau est livré un lot de configurations par défaut appelée *defconfig* qui, dans le cas qui nous intéresse ici, se trouve dans le sous-répertoire `arch/arm/configs`. On y trouve, entre autres le fichier `bcmrpi_defconfig` pour la Raspberry Pi ou `bcm2709_defconfig` pour la Raspberry Pi 2.

Ce *defconfig* est utilisable pour configurer les sources à l'aide d'un simple `make bcmrpi_defconfig` par exemple ou plus exactement un `make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- bcmrpi_defconfig`. Les deux options passées en argument permettant respectivement :

- de préciser l'architecture (`ARCH`) ;
- et de spécifier la « racine » de la chaîne de compilation à utiliser (`CROSS_COMPILE`) puisque nous avons l'intention de compiler un noyau pour une autre plateforme que celle actuellement utilisée.

*ancienne expression terrienne, pleine de puissance, de sagesse et de consolation pour les âmes dans le besoin



Si vous n'êtes pas à l'aise avec les interfaces de type texte, le noyau propose également quelque chose de plus graphique en termes de gestion de la configuration des sources, via une « make xconfig ». Il vous faudra cependant installer les paquets de développement de Qt.

Notez que ces deux options (qui sont en fait des déclarations de variables utilisées par `make`) ne sont pas systématiquement nécessaires, mais qu'il est de bon ton de toujours les spécifier, ne serait-ce que pour simplement éviter de les oublier lorsqu'elles sont indispensables (compilation).

Après le `make bcmrpi_defconfig`, les sources seront configurées grâce au `.config` à la racine des sources. Vous pourrez utiliser `make menuconfig` pour apporter des modifications. `menuconfig` est seulement l'une des cibles utilisables et propose une interface textuelle permettant de naviguer dans les options (en cas de problème, installez le paquet `libncurses5-dev`). Si vous préférez une interface graphique, optez pour `make xconfig`. Et pour une solution sous forme de questions/réponses presque inutilisable, ce sera `make config` (cette cible peut être vue comme la solution historique).

Mais il existe une solution bien plus efficace que la configuration par défaut pour une plateforme : celle consistant à utiliser une configuration déjà en place et connue pour fonctionner. Votre carte Raspberry Pi et son système Raspbian disposent d'un noyau fonctionnel et ce noyau embarque normalement une configuration. Pour y accéder, rien de plus simple, on commence par charger le module `configs` :

```
$ sudo modprobe configs
```

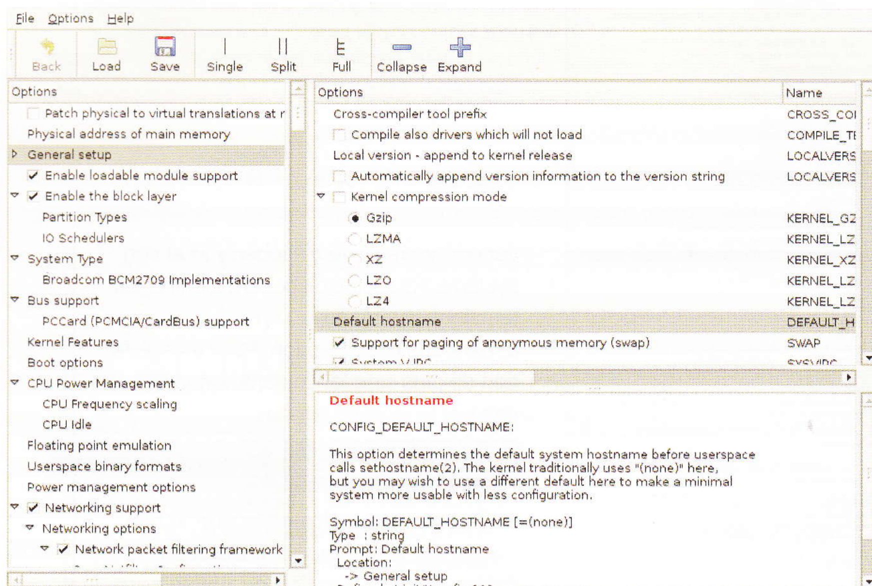
Dès lors, vous trouverez un fichier `config.gz` dans `/proc`. Il s'agit du fichier de configuration du noyau compressé avec Gzip. Vous pouvez en visualiser le contenu d'un simple `zcat /proc/config.gz` (ou `zless`) :

```
#
# Automatically generated file; DO NOT EDIT.
# Linux/arm 4.1.8 Kernel Configuration
#
CONFIG_ARM=y
CONFIG_SYS_SUPPORTS_APM_EMULATION=y
CONFIG_HAVE_PROC_CPU=y
CONFIG_STACKTRACE_SUPPORT=y
CONFIG_LOCKDEP_SUPPORT=y
CONFIG_TRACE_IRQFLAGS_SUPPORT=y
CONFIG_RWSEM_XCHGADD_ALGORITHM=y
[...]
```

La structure du fichier est simplissime puisqu'il ne s'agit que d'une liste affublée d'un `y` pour *yes* ou d'un `m` pour module. Les options ou pilotes non actifs sont placés en commentaire (précédés d'un `#`) et complétés d'un `is not set`. Comme le précise le message en début de fichier, cette configuration n'est pas destinée à être éditée/modifiée à la main.

Pour l'utiliser avec vos sources du noyau, vous devez transférer le fichier sur la machine de construction (PC) et le décompresser (`gunzip config.gs`) avant de le placer à la racine des sources en l'appelant `.config`. Là, utilisez la commande `make oldconfig` :

```
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- oldconfig
  HOSTCC  scripts/basic/fixdep
  HOSTCC  scripts/kconfig/conf.o
  SHIPPED scripts/kconfig/zconf.tab.c
  SHIPPED scripts/kconfig/zconf.lex.c
  SHIPPED scripts/kconfig/zconf.hash.c
  HOSTCC  scripts/kconfig/zconf.tab.o
  HOSTLD  scripts/kconfig/conf
scripts/kconfig/conf  --oldconfig Kconfig
#
# configuration written to .config
#
```



Cette cible de `make` va lire le fichier et éventuellement demander s'il faut ou non activer des fonctionnalités qui sont disponibles dans les sources, mais non présentes dans la configuration « importée ». Si la version du noyau utilisée sur la machine d'origine n'est pas trop distante de celle des sources, il est fort probable qu'aucune question n'apparaisse. De plus, même en cas de réponse erronée de votre part, les changements peuvent ensuite être ajustés via un `make menuconfig`.

Si vous n'aimez pas Qt ou n'avez pas envie d'installer les paquets supplémentaires liés, peut-être souhaiterez-vous reposer sur GTK+ et Glade. Voici « make gconfig » : mêmes données, même configuration, différente interface.

La configuration du noyau dépendra ensuite de la raison pour laquelle vous souhaitez en compiler un nouveau : activer des options de débogage ou un pilote spécifique, changer quelques paramètres, intégrer vos corrections et modifications, etc.

Une petite remarque tout de même : allez faire un tour dans la configuration, dans ***General setup*** puis ***Local version - append to kernel release***. Vous pouvez préciser ici un « complément » qui s'ajoutera au numéro de version. Le numéro de version est utilisé, entre autres, pour composer le nom de répertoire dans lequel sont placés les modules. En personnalisant cette information, vous vous assurerez de ne pas rentrer en conflit avec une autre instance du noyau et des modules, dans une version identique. N'utilisez pas en revanche de majuscules ! Ceci fonctionne très bien lors de la construction, mais sera incompatible avec la dernière option que nous envisagerons en fin d'article (construction de paquets).

Enfin, même si cela peut paraître logique, ajoutons qu'un noyau compilé pour Raspberry Pi 2B ne fonctionnera pas sur Raspberry Pi B+ par exemple. Avec la distribution Raspbian à jour, vous pourrez remarquer que sur la 2B nous avons :

```
$ uname -a
Linux raspberrypi 4.1.7-v7+ #817 SMP PREEMPT
Sat Sep 19 15:32:00 BST 2015 armv7l GNU/Linux
```

et sur la B+ :

```
$ uname -a
Linux rpiaplus 4.1.7+ #817 PREEMPT
Sat Sep 19 15:25:36 BST 2015 armv6l GNU/Linux
```

La version du noyau est identique (4.1.7), mais le fameux « *local version* » est différent. « + » sur la B+ et « v7+ » sur la 2B. Le « v7 » fait référence à l'architecture « armv7l » de la Raspberry Pi 2 (cf. plus loin dans l'article). Il ne faut donc pas prendre le `/proc/config.gz` de l'une pour compiler un noyau destiné à l'autre. La carte ne démarrera tout simplement pas.

Une simple comparaison des deux `/proc/config.gz` montre clairement pourquoi :

```
grep ARCH_BCM 2B_config Bp_config
2B_config:# CONFIG_ARCH_BCM2708 is not set
2B_config:CONFIG_ARCH_BCM2709=y
Bp_config:CONFIG_ARCH_BCM2708=y
Bp_config:# CONFIG_ARCH_BCM2709 is not set
```

2.3 Construction

La compilation ou construction en tant que telle n'est guère compliquée, vous avez déjà fait la plus grande part du travail. Une simple commande suffira pour produire ce que vous souhaitez : `make`. Bien sûr, ceci n'est valable que si vous comptez produire un noyau pour une plateforme identique à l'hôte actuel. Dans notre cas, ce sera plutôt (ne le faites pas de suite) :

```
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-
```

La compilation d'une telle masse de sources est une tâche gourmande en mémoire et en ressources processeur et, par défaut, un seul *thread* sera utilisé. Bien entendu, si vous disposez d'une machine avec plusieurs cœurs, ce serait un beau gâchis. Pour accélérer tout cela, on utilise l'option `-j` suivie du nombre de tâches à exécuter en parallèle. Ce qui nous donnera par exemple sur une machine plus ou moins standard : `make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- -j 9` (4 cœurs, deux threads par cœurs, plus un).

Au terme de la longue compilation, l'ensemble du noyau et des modules seront produits, normalement sans erreur. En cas de problème cependant, la première chose à faire sera de réitérer la commande en réduisant le nombre de tâches, voire en supprimant l'option `-j`.

Mais si tout se passe correctement, l'image du noyau sera créée et placée dans le fichier `arch/arm/boot/zImage` (le même format que celui du `kernel.img` présent sur la carte SD/microSD), mais ce n'est pas tout. Les modules noyau (`*.ko`) seront également compilés, mais se trouveront disséminés un peu partout dans les sources.

Pour produire l'arborescence adéquate (qui ira dans `/lib/modules`), il existe une cible spécifique, `modules_install` provoquant, par défaut, l'installation des modules sur la machine courante. Nous voulons ici plutôt obtenir un équivalent dans un répertoire donné. Nous spécifions donc une destination via l'option `INSTALL_MOD_PATH` :

```
$ mkdir ~/racine_pi
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- \
INSTALL_MOD_PATH=~/racine_pi modules_install
```

Nous choisissons ici de créer un `racine_pi` dans le répertoire personnel de l'utilisateur courant. Ce répertoire fera alors office de racine du système de fichiers. Pour installer les modules, il suffira de copier tout son contenu à la racine `/` de la Raspberry Pi. Le noyau lui-même, ou plus précisément l'image du noyau, devra être copiée sur la partition FAT de la carte SD (représentée par `/boot` dans le système). Deux options s'offrent alors à vous, soit vous conservez une copie de `kernel.img` et écrasez l'original avec `arch/arm/boot/zImage`, soit vous lui placez simplement le fichier sur la carte et ajoutez une ligne dans le `config.txt` :

```
kernel=zImage
```

Au prochain démarrage, la Raspberry Pi va charger ce nouveau noyau et, si tout se passe bien, le démarrage poursuivra son cours sans le moindre problème. Vous aurez alors la satisfaction de vérifier que vous utilisez effectivement votre tout nouveau noyau compilé avec vos petites mimines (et celles de votre PC surtout) via un petit `uname -a`. Bravo !

2.4 Et pourquoi pas un paquet ?

Copier à la main un noyau et toute une arborescence de modules fonctionne, mais n'est pas nécessairement la solution la plus élégante. N'oubliez pas que Raspbian est une distribution GNU/Linux Debian adaptée à Raspberry Pi. Et qui dit « distribution » dit « gestion de paquets ». Comble du bonheur, les sources du noyau comprennent des cibles permettant de produire directement des paquets :

- `binrpm-pkg` pour les systèmes basés sur RPM comme Fedora ;
- `tar-pkg` pour une simple archive Tar ;
- `targz-pkg` pour une archive Tar compressée avec Gzip ;
- `tarbz2-pkg` idem, mais compressée avec Bzip2 ;
- et surtout `deb-pkg` pour créer des paquets Debian (ou Ubuntu).

Il vous suffira donc d'un simple `make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- deb-pkg` pour procéder à la compilation

enchaînant sur la création des paquets. Cette étape sera bien plus rapide si vous avez déjà procédé à un simple `make` au préalable. Ce qui est généralement une bonne idée, ne serait-ce que pour vérifier que tout se déroule convenablement.

Notez que, par défaut, ce seront des paquets pour l'architecture armel qui seront produits, ce qui fonctionnerait sur des Raspberry Pi A, B, A+ et B+ équipés d'un système Raspbian ancien ou d'une distribution Debian armel. Le système Raspbian actuel sur tous les modèles de Raspberry Pi (ARMv6 et ARMv7 pour la Pi 2) utilise maintenant l'architecture armhf.

Pour la petite histoire, le processeur des Raspberry Pi (non 2) possède une architecture ARMv6 ISA avec VFP2, ce qui pour le projet Debian est insuffisant pour être éligible au port armhf de la distribution (minimum ARMv7 avec VFP3). Normalement, ces modèles de Pi devraient être pris en charge dans le même port que armv4t, Debian armel, et donc avec des opérations en virgule flottante relativement lentes. Les modèles A, B, A+ et B+ se trouvent donc, en quelque sorte, entre deux architectures, armel pour (la vraie) Debian, mais armhf pour Raspbian.

Le problème ne se pose pas vraiment avec Raspbian, mais le système de construction de paquets intégré dans le noyau, lui, n'en a pas connaissance. Pour corriger le problème, nous devons utiliser un argument supplémentaire avec `make` : `KBUILD_DEBARCH=armhf`. Ce qui nous donnera :

```
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- \
KBUILD_DEBARCH=armhf deb-pkg
```

Notez que si vous avez précisé un complément de version en majuscule, la compilation se déroulera correctement, mais la création de paquet échouera, car les majuscules ne sont pas autorisées à ce niveau par la politique de nommage Debian. À l'issue de l'exécution de la commande, vous devrez trouver les paquets suivants dans le répertoire parent :

- `linux-image-4.1.8-lef+_4.1.8-lef+-1_armhf.deb` : image du noyau et modules ;
- `linux-firmware-image-4.1.8-lef+_4.1.8-lef+-1_armhf.deb` : les images de firmware pour des périphériques spécifiques (ce ne sont pas les mêmes que ceux mis à disposition sous forme de paquets individuels, comme `firmware-ralink` par exemple) ;
- `linux-headers-4.1.8-lef+_4.1.8-lef+-1_armhf.deb` : les fichiers d'entêtes nécessaires au développement noyau ;
- `linux-libc-dev_4.1.8-lef+-1_armhf.deb` : les fichiers d'entêtes fournis par le noyau pour le développement d'applications utilisateur.

Il vous suffit alors de copier tout cela sur votre Raspberry Pi et de procéder à l'installation avec `sudo dpkg -i` suivi de la liste des noms de fichiers. L'image du noyau sera installée dans `/boot` avec un nom dépendant de la version du noyau et du complément de version que vous aurez précisé (ici `/boot/vmlinuz-4.1.8-lef+`. Il vous suffira alors de changer votre `config.txt` pour utiliser ce noyau (directive `kernel=`) et redémarrer votre carte.

CONCLUSION

Cet article est finalement bien plus long que je m'y attendais, mais nous avons fait le tour des options et solutions les plus « propres » pour procéder à une recompilation du noyau si d'aventure cela vous serait nécessaire. Pour terminer, je préciserai que le travail de l'équipe Raspbian et de la fondation concernant la fourniture d'un noyau le plus complet et le plus stable possible est remarquable. Cependant, l'ensemble des fonctionnalités et des pilotes ne sont pas activés. Un exemple simple concerne le capteur de pression BMP085 en SPI ou i2c (voir *Hackable n°2*), mais absent du noyau Raspbian. Il en va de même pour la plupart des contrôleurs de leds (comme le fantastique PCA9532). Quoi qu'il en soit, en explorant tout cela, gardez à l'esprit que vous vous trouvez dans un territoire qui n'est pas forcément celui où tout le monde joue et qu'il est judicieux de se ménager des solutions de repli, faire des sauvegardes et rester attentif et prudent. DB